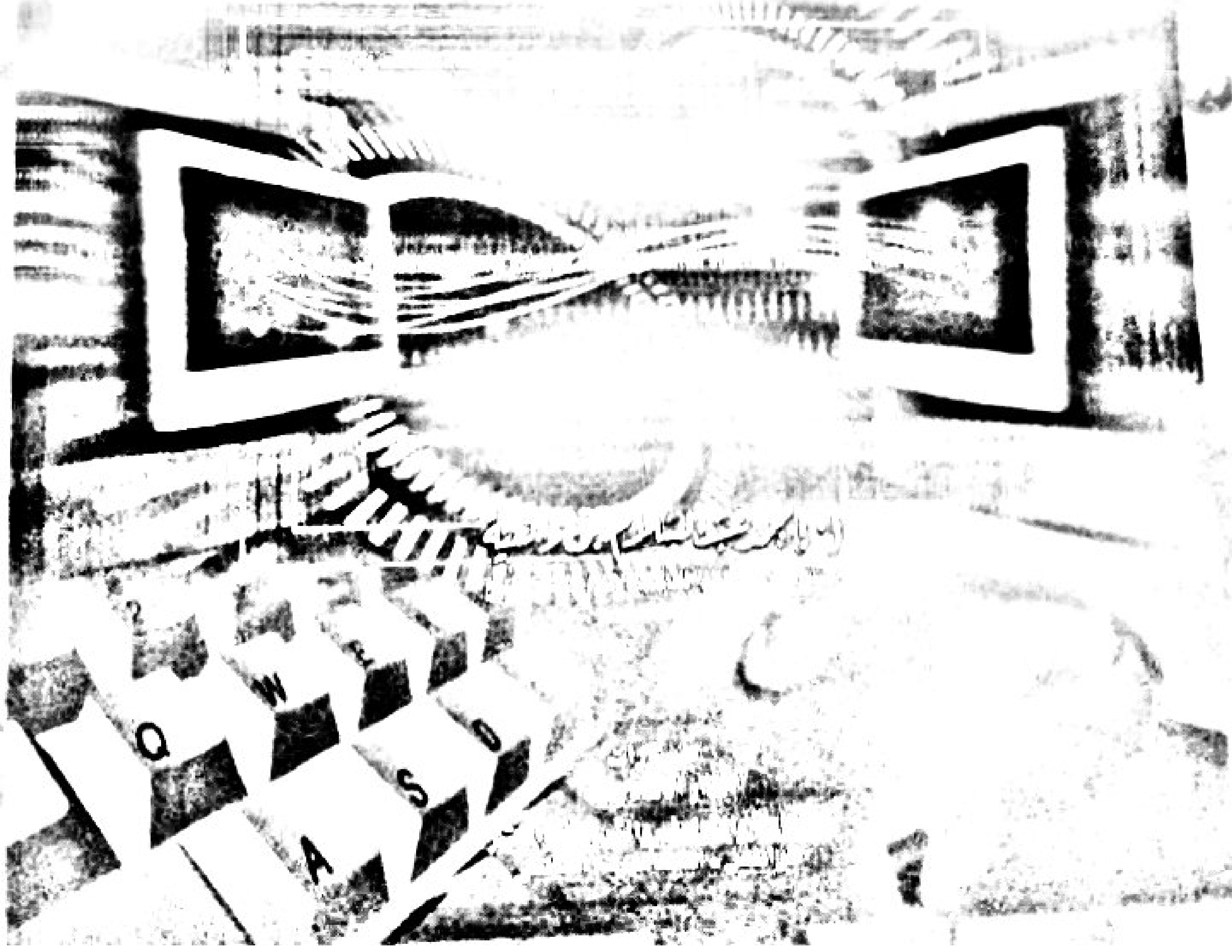
هندسة البرمجيات

Software Engineering



المحتويات

قائمة أشكال الكتاب

تمهيد

لقد تعايش الإنسان مع البرمجيات Software خلال الخمسين سنة الماضية بحلوها ومرّها حيث أصبحت تعيش معه يوميا، فهي معه في منزله على شكل ألعاب الحاسوب أو برامج ثقافية، وفي مكتبه على شكل برامج طباعة أو إعداد جداول، وفي تسوقه من خلال منظومة المبيعات.

وعلى نطاق أوسع يجد أن مرتبه يتم حسابه عن طريق البرمجيات، وإذا أراد حجز مقعد في طائرة فمن المؤكد ان ذلك سيتم عبر برمجية الحجز. وربما يجد في المصحّات أن بياناته الشخصية والصحية قد تمّ تخزينها في ملفات على أقراص صغيرة الحجم داخل أو خارج جهاز الحاسوب.

وبعبارة أخرى سيجد أن البرمجيات تؤدي له الخدمات في شتىّ النشاطات الاقتصادية والاجتماعية والترفيهية.

والسؤال الآن هو كيف يتم أداء هذه البرمجيات بحيث تكون خالية من الأخطاء، علما بأنها تكون في منتهى التعقيد والتشعب؟

يجب أن نلاحظ أن إعداد البرمجيات يشبه إلى حد كبير بناء الجسور والعمارات من حيث ضرورة التصميم والتنفيذ والصيانة من قبل البشر باستخدام طرق وأدوات ومنهجيات تمّ إعدادها من قبل خبراء متخصصين. والفرق الوحيد بينهما أن البرمجيات غير منظورة بينما الجسور والعمارات بالامكان إدراكها وتحسسها.

إن طريقة إعداد وصيانة البرمجيات ليست بالعملية السهلة، فهي يمكن أن تصبح مملة ومتعبة نظرا لتعقد التطبيقات وزيادة احتياجات Needs ومتطلبات Requirement الزبائن المتنوعة والتي قد لا تنتهي.

وقد ينعكس ذلك على حدوث مشاكل تخص الجودة وزمن وتكاليف المنظومة المراد إعدادها.

لهذه الأسباب أصبح لزاما و ضروريا أن يتم اتباع أسلوب سليم ومنهجية فائقة تصل بنا إلى الهدف المنشود، و هذا الأسلوب يسمى بالأسلوب الهندسي Engineering Discipline في إعداد البرمجيات أو بما يسمى بعلم هندسة البرمجيات Software Engineering وهو علم يبحث في الأساليب الصحيحة لإعداد البرمجيات بأقل التكاليف ولكن بفاعلية وجودة عالية .

وهذا الكتاب يتعلق بعلم هندسة البرمجيات الذي يتم فيه تطبيق الأسلوب السليم لتحليل وتصميم وتنفيذ واختبار وصيانة أي برمجية مهما كان حجمها وتعقيدها. وقد أصبح هذا المجال ضروريا للمتخصصين في إعداد البرمجيات خاصة بعد الثورة الجديدة في تقنية المعلومات Information Technology والانترنت Internet و الاتصالات عن بعد Telecommunications والوسائط المتعددةMultimedia.

لقد أضحى مجال هندسة البرمجيات مهما للغاية فتغيرت مناهج أقسام الحاسوب في الجامعات والمعاهد لتشمل هذا المقرر، وصارت مهنة مهندس البرمجيات Software Engineer من المهن المرغوبة و التي تجلب عائدا ماليا لصاحبها.

وبالرغم من مرور أكثر من أربعين سنة على ظهور مجال هندسة البرمجيات ورغم التحسن الواضح في البرمجيات التي تم إعدادها في الآونة الأخيرة، إلاّ أنه لا تزال هناك بعض العوائق حيث تواجه الشركات صعوبات في إعداد برمجيات ذات كفاءة عالية و خالية من العيوب .

كما نجد الكثير من المبرجين المبتدئين ليسوا على دراية بمستجدات هذا العلم الذي تطور يوما بعد يوم مما أوجد فجوة بين خريجي العهود الأولى من انطلاقة علم الحاسوب والجيل الجديد (جيل هندسة البرمجيات).

لقد حاولت في هذا الكتاب قدر المستطاع أن يكون مبسطا بما يكفي بحيث تصبح مادة الموضوع في حدود ما يكفي لمقرر واحد في مستوى التدريس الجامعي أو يزيد عن ذلك قليلا.

وقد قسمّت هذا الكتاب إلى أحد عشر فصلا في هذا العلم.

الفصل الأول يتعلق بالبرمجيات وهندستها و نظمها، وهذا يعطي صورة واضحة على

معنى هندسة البرمجيات بالمقارنة بعتاد الحاسوب وهندسته، ثمّ تعرضت في الفصل الثاني إلى أهم نشاطات التخطيط Planning وهي جدولة المشروع البرمجي وتقدير التكاليف، وهذان العنصران مهمان ويجب أخذهما في الاعتبار في أي مشروع برمجي Software Project.

وفي الفصل الثالث تعرضت إلى أهم عناصر المشروع البرمجي وهو تحديد المتطلبات ومشاكلها وكذلك سبل الوصول إلى متطلبات كاملة ودقيقة.

وفي الفصل الرابع و الخامس تناولت موضوع طرق و أدوات تصميم البرمجيات.

أمّا في الفصل السادس فقد تناولت موضوع لغات البرمجة وتطورها و مساهمتها في الحصول على أعلى جودة وبأقل التكاليف و خلال مدة زمنية قصيرة.

كما تطرقت إلى التقنية الشيئية وبعض أدواتها فهي تعتبر لغة العصر في مجال البرمجة.

ولأن الجودة Quality هي مقياس نجاح أو فشل أي مشروع برمجي فمن الأهمية بمكان أن يتناول الكتاب هذا الموضوع، حيث خصصت الفصل السابع ليتناول مشكلة تأكيد الجودة وأنواع الاختبارات المراد أداؤها للوصول إلى هذا الهدف.

وفي الفصل الثامن يتناول الكتاب موضوع النماذج المتعددة في عملية إعداد البرمجيات Software Development Models.

أمّا الفصل التاسع فيتناول موضوع إدارة المشروع البرمجيSoftware Project Management التي بدونها لن ينجح المشروع، والفصل العاشر يتناول تأثير العنصر البشري في إعداد المنظومة، والفصل الحادي عشر يتناول أهم الأدوات المستخدمة في إعداد وصيانة المنظومات مثل أدوات (CASE) .

وأخيرا نأمل أن يكون هذا الكتاب دليلا مساعدا لطلاب علم الحاسوب وتقنية المعلومات خاصة في مقررات هندسة البرمجيات وتحليل النظم وإدارة المشاريع البرمجية، وأن يكون لهم عونا في إعداد مشاريع التخرج في الجامعات والمعاهد العليا، وأن يكون مرجعا لطلاب الدراسات العليا، وأن يسدّ فراغا في المكتبة العربية التي تفتقر إلى أي مساهمة فاعلة في هذا الفرع من علوم المعرفة.

أشكر الله سبحانه وتعالى الذي منحني الإرادة والصحة والصبر للبدء والاستمرار في إعداده إلى نهايته وما التوفيق الاّ بالله.

كما أشكر في هذه المقدمة زميلي الاستاد الدكتور عمر محمد زرتي عضو هيئة التدريس بقسم الحاسوب كلية العلوم بجامعة الفاتح على مساعدته القيّمة ومراجعته العلمية طيلة مدة الكتابة، وكان لتشجيعه المستمر حافزا لي للوصول إلى نهاية مسيرة إعداد الكتاب التي لم تكن سهلة.

كما أشكر المهندس زهير محمد القماطي على مساعدته في عملية الإخراج الفني بشيء من الإبداع والابتكار.

ولا يفوتني أن أشكر الدكتور محمد عبدالله القماطي على قيامه بمراجعة الكتاب باللغة العربية.

لقد فاز هذا الكتاب بالترتيب الثاني في مسابقة الكتاب التي أجرتها اكاديمية الدراسات العليا بطرابلس، ولا يسعني هنا إلاّ أن أتقدم للجنة على هذه المسابقة بالشكر والامتنان على حسن تقديرهم للجهد الذي بذلته في إعداد هذا الكتاب، وبذله المراجع في تدقيقه.

ونسأل الله القدير أن يكون هذا الكتاب في ميزان حسناتنا جميعا، وإن يزيدنا من فضله، إنّه ذو الفضل العظيم..

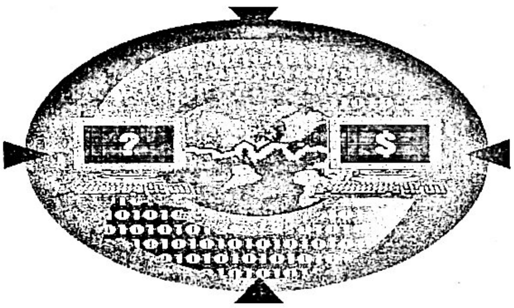
المؤلف

طرابلس 2009/8/8

الفصل الأول

هندسة عملية إعداد البرمجيات

Engineering the Software Development Process



1.1 مقدمة

يستخدم مصطلح برمجيات software عادة للإشارة إلى برامج الحاسوب Computer Programs.

وفي الحقيقة ليس هذا إلا جزءا من التعريف حيث تتكون البرمجيات بصورة عامة من:

- برامج Programs: تعليمات للحاسوب لتنفيذ عمل محدد.
- التوثيق Documentation: وثائق تصف طريقة إعداد البرنامج واستخدامه و من أمثلتها كتيب التشغيلUser Manual .
- البيانات Data: وتكون إمّا على صورة نص أو صوت أو صورة ثابتة أو متحركة (فيديو).

في البداية كانت كلمة (برنامج) هي المستخدمة فيما يعرف اليوم بالبرمجيات software (المنظومات) بعد اضافة التوثيق.

ولا شك أن نقص التوثيق أو عدم وجوده يؤدي إلى الإرتباك في متابعة المنظومة وتشغيلها من قبل المستخدم بصورة مثلى.

لذلك بدأ الاهتمام بالتوثيق ولكن لم يخل الأمر من بعض المشاكل حتى بعد إضافة التوثيق، فظهرت مشاكل متعددة بسبب تنوع وتعقد المنظومات المراد إعدادها.

هذه المشاكل عرفت باسم أزمة البرمجيات وهي تعني المشاكل التي تمت مواجهتها أثناء إعداد المنظومات البرمجية، نظرا لزيادة احتياجات ومتطلبات الزبون وازدياد حجم البرمجيات.

لذلك التجأ المختصون للبحث عن حل لهذه الأزمة، وكان الحل هو اتباع الأسلوب الهندسي في إعداد المنتوج على شكل مشروع، مما يستوجب تعيين مدير مشروع وفريق عمل.

والنقطة الأخرى في الأسلوب الهندسي هي أن المنتوج يمر عبر مراحل هي: التحليل، والتصميم، والتنفيذ، والإختبار، والصيانة.

والنقطة الثالثة التي استدعت اللجوء إلى الأسلوب الهندسي هي الاهتمام بجودة المنتوج، ولا شك أن موضوع الجودة يعتبر من أولويات الاهتمامات.

و في هذا الباب سنتطرق إلى موضوع الأسلوب الهندسي في إعداد المنظومات وهو موضوع هذا الكتاب.

ما هي خصائص برمجيات كل جيل؟

1.2 تاريخ البرمجيات History of Software

مرت برمجة الحاسوب بخمسة عصور لتصل إلى ما وصلت إليه اليوم.

وفيما يلي التقسيم لهذه الحقبات التي تسمى عادة بأجيال الحاسوب:

الجيل الأول: The First Era (1965-1946):

تميز هذا الجيل بما يعرف بالمعالجة بالدفعة Batch Processing حيث كانت البرامج تجمع في مجموعات (دفعات) لغرض المعالجة الجماعية، كما كان التوزيع محدودا Limited Distribution فلم تكن أغلب الشركات تهتم بالحاسوب وتطبيقاته وكان البرنامج الواحد مخصصا لمستعمل واحد وهو ما يعرف بالبرمجيات المخصصة Custom Software حيث نجد أن كل برنامج مصمم لزبون معين، وكانت معظم البرمجيات ذات طابع علمي وهو ما يعرفّ الآن بالبرمجيات العلمية.

الجيل الثاني: The Second Era (1975-1966)

في هذا الجيل أصبح الحاسوب قادرا على أن يخدم أكثر من مستعمل في وقت واحد، وهو ما يعرف عادة بنظام تعدد المستعملين Multi-user system وقد نتج عن هذا النظام أسلوب جديد في البرمجة) البرمجة المتعددة (Multiprogramming حيث أصبح بالإمكان تنفيذ عدة برامج في نفس الوقت، وتطورت مهام أنظمة التشغيل لتشمل المعالجة متعددة الوظائف Multitask Processing التي تسمح لأكثر من برنامج واحد أن يبقى في الذاكرة في نفس الوقت.

كما تميز هذا العصر بالمعالجة التفاعلية Interactive Processing التي مكنّت المستعمل من الاتصال مباشرة بالحاسوب وأن يتجاوب مباشرة مع النتائج في نفس الجلسة.

إلى جانب ذلك توفرت المعالجة بالوقت الحقيقي Real-time Processing وهي ما يعرف عادة بالمعالجة المباشرة On Lineمع وقت استجابة قصير وترجيع سريع.

وتطورت البرمجيات وأصبحت تعتبر منتوجا تجاريا، وبدأ ظهور مؤسسات البرمجة، ورافق ذلك ظهور الجيل الأول من نظم إدارة قواعد البيانات Data Base Management Systems (DBMS).

وخلال هذه الفترة ظهرت أزمة البرمجيات Software Crisis وبدأ هذا المصطلح في التداول بين المختصين في البرمجيات ليعبر عن المشاكل التي كانت تواجههم أثناء إعداد واستخدام المنتوج البرمجي.

الجيل الثالث: The Third Era (1985- 1976)

تميز هذا الجيل بما يلي:

- النظم الموزعة Distributed Systems وهو نظام توجد فيه عدة حواسيب تعمل معا في نفس الوقت بطريقة منسقة.
- هبوط في أسعار المعدات Low cost hardware لتصبح رخيصة الثمن في متناول العديد من الشركات والأفراد.
- أزمة البرمجيات ازدادت مع تنوع التطبيقات وأصبح البحث عن حل لهذه الأزمة ضروريا.
- استعمال المعالج الدقيق microprocessor في كثير من التطبيقات مثل:
  - السيارات Automobiles.
  - أفران الموجات الدقيقة Microwave ovens.
  - الإنسان الآلي Robot.

- محطات الطاقة Power Plants.

- التحكم في الحركة الجوية Air traffic Control.

- استعمال الحواسيب الدقيقة Microcomputers.

حيث أصبح الحاسوب في متناول الكثير من عامة الناس.

الجيل الرابع: (1986 - 1994) The Fourth Era

تميز هذا الجيل بما يلي:

- ■ أنظمة تشغيل الحواسيب الشخصية PC operating systems مثل: DOS وWINDOWS.
- ■ حواسيب صغيرة وقوية Powerful Microcomputers.
- ■ شبكات الحاسوب المحلية والعالمية Local / global networks.

الجيل الخامس: (1995 - ؟)

تميز هذا الجيل بما يلي:

- ■ اتساع تطبيقات الشبكات المحلية والواسعة Networks (Wan/LAN) واستعمال الانترنت internet على نطاق واسع.
- ■ الاهتمام المتزايد بمفهوم الذكاء الاصطناعي Artificial intelligence والنظم الخبيرة Expert systems.
- ■ التقنيات الشيئية Object Oriented technology.
- ■ البرمجة الجزئية Component programming.
- ■ إعادة استعمال البرمجيات Software reuse.
- ■ إنجاز العديد من التطويرات مثل المعالجة المتوازية parallel processing والأقراص البصرية optical disks والوسائط المتعددة multi-media.

ملاحظة: تواريخ العصور أو الأجيال تقريبية وتختلف من كتاب إلى آخر

1.3 أزمة البرمجيات Software Crisis

إن الانتشار الواسع في استعمال الحاسوب جعل المجتمع يعتمد بصورة متزايدة على البرمجيات.

ومن أمثلة هذه البرمجيات أو المنظومات البرمجية التطبيقية مايلي :

- ■ النظم المصرفية Banking system.
- ■ سوق البورصة Stock market.
- ■ حجز رحلات الخطوط الجوية Airline reservation.

هذه النظم وغيرها لا تتسامح مع مشاكل البرمجيات والخطأ مكلف للغاية، لذلك ونظرا للعديد من المشاكل المتعلقة بإعداد البرمجيات، فقد نشأت في عالم الحاسوب أزمة تسمى بأزمة البرمجيات. فرغم التقدم الكبير في إعداد البرمجيات إلا أن كثيرا من الصعوبات لا زالت موجودة وتشكل إعاقة لعملية إعداد برمجيات حاسوب عالية الجودة.

و المشاكل التي غالبا ما كانت وراء أزمة البرمجيات هي:

- ■ الجدول الزمني وتقدير التكاليف غير دقيق حيث يصل التأخير إلى أشهر أو سنوات.
- ■ جودة البرمجيات أحيانا كانت أقل من المستوى المطلوب.
- ■ متطلبات المستخدم أحيانا لا تتحقق.
- ■ إنتاجية المبرمجين لم تواكب الازدياد المستمر في طلب الحاجات، والزبائن لم يكونوا دائما راضين عن البرمجيات الجديدة بسبب معدل الأخطاء.
- ■ لم يأخذ منتجو البرمجيات الوقت الكافي لجمع البيانات والمتطلبات ذات العلاقة بعملية الإنتاج.
- ■ البرمجيات المتوفرة كانت صعبة الصيانة بدرجة كبيرة.
- ■ البرمجيات التي تم تسليمها كانت تأتي متأخرة، وبتكلفة أعلى من المتوقع. (هذه المشكلة قد تكون ذات علاقة بالمشكلة الأولى).

- البرمجيات المتوفرة لم يتم اختبارها في الأغلب بشكل صحيح وبشكل كامل وبالتالي فهي غير موثوق بها.

ما هي أسباب الأزمة ؟

تتمثل أسباب الأزمة فيما يأتي:

1. عدم استقرار متطلبات المستخدم
2. تغير المتطلبات Requirement change بصورة أسرع مما يجب.
3. أدوات العتاد والبرمجيات تتغير بمعدل أسرع .
4. بعض المدراء المسؤولين عن إعداد البرمجيات ليس لديهم خلفية كافية عن منهجيات إعداد البرمجيات (المنظومات).
5. نقص التواصل بين معدي البرامج والزبائن.
6. التقدير السيئ للإمكانيات.
7. التخطيط السيئ للمشروع
8. الطرق الضعيفة لقياس جودة البرمجيات.
9. الأهداف والغايات ذات تعريف غامض.
10. عدم التوافق مع المعايير القياسية.
11. العوامل الإنسانية مثل المقاومة الطبيعية عند الإنسان لأي تغيير.

ملاحظة: في هذا الكتاب عادة مانستخدم مصطلح البرمجيات بمعنى منظومة تطبيقية أو منتوج برمجي.

1.4 حقائق ومفاهيم خاطئة Facts & Misconceptions

إلى جانب أسباب أزمة البرمجيات كان هناك خرافات (مفاهيم خاطئة) سببت في الربكة والعوائق في إعداد البرمجيات.

الخرافة البرمجية هي مفهوم خاطئ ومنتشر يعتقد به كثير من المدراء والمستخدمين ومعدّي البرمجيات، وهذه الخرافات هي بقايا من الجيل الأول لإعداد البرمجيات.

وتتلخص هذه المفاهيم الخاطئة في الآتي :

المفهوم الخاطيء الأول: يوجد الكثير من الكتب تعطي منهجيات وأدوات ومعايير لإعداد البرمجيات.

الحقيقة: قد يوجد بعض الكتب ولكن هل يتم استعمالها؟ هل معدو البرامج مدركون لوجودها؟ هل هي كاملة؟ عادة ما تكون الإجابة «لا».

المفهوم الخاطيء الثاني: عند التأخر في الجدول الزمني يمكننا إضافة مبرمجين جدد لإنجاز المشروع في الوقت المناسب.

الحقيقة: إعداد البرمجيات ليس عملية آلية، فمن المتعارف عليه في هندسة البرمجيات أن إضافة مبرمجين جدد لمشروع متأخر يزيد من مشكلة التأخير والسبب في ذلك أن مشكلة التواصل بين أعضاء فريق إعداد المنظومة تزيد سوءا كلما زاد عدد أعضاء الفريق، ويمكن زيادة هذا العدد ولكن بطريقة مخطط لها بدقة.

كما انها تسببت في مشكلة ثانية هي الحاجة لتدريب الأعضاء الجدد لفريق المنظومة فمثلا يجب تدريب المبرمجين الجدد قبل دخولهم للمشروع وذلك يتطلب وقتا محسوبا ومضافا للجدول الزمني.

المفهوم الخاطيء الثالث: تعتبر المذكرة العامة عن أهداف ومتطلبات (المنظومة) المشروع كافية للبدء في إعداد المنظومة، حيث يمكن إضافة التفاصيل فيما بعد.

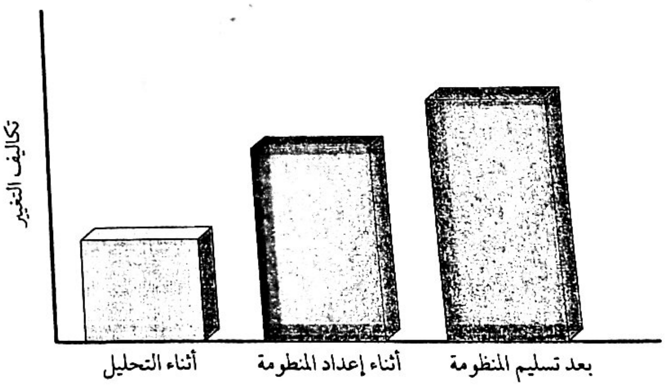
الحقيقة: التعريف المقتضب للمتطلبات هو السبب الرئيسي للمجهودات الفاشلة في إعداد البرمجيات حيث من المهم أن يحتوي التعريف على وصف مفصل للمتطلبات التي منها :

- وظائف البرمجيات.
- الأداء.
- واجهة المستخدم.
- قيود التصميم.
- معيار الجودة.

المفهوم الخاطىء الرابع: يمكن دائما التغلب على مشكلة تغير متطلبات المشروع لأن البرمجيات ذات مرونة.

الحقيقة: تأثير تغير المتطلبات يتغير مع المراحل الزمنية، فإذا حدث التغيير في المراحل مثل مرحلة التحليل فإن هذا التغيير يمكن أن ينجز بسهولة، ولكن إذا حدث التغيير في مرحلة متأخرة من المشروع البرمجي فإن التكلفة تزيد بصورة كبيرة.

يبين الشكل (1.4.1) تقريبيا تكلفة التغييرCost of Change في المراحل المختلفة لعملية إعداد البرمجيات.



الشكل (1.4.1) تكلفة المشروع

المفهوم الخاطئ الخامس: كلما أسرعت في البدء بكتابة شفرة البرمجة أنهيت المشروع في وقت أقل.

الحقيقة: هذا المفهوم صحيح في حالة البرامج الصغيرة، ولكن في المسائل المعقدة كلما أسرعت في كتابة شفرة البرنامج أبطأت في إنهائه بصورة صحيحة، وإذا لم تأخذ وقتا كافيا في مراحل ما قبل كتابة البرنامج فإنك ستدفع الثمن فيما بعد في ضياع الوقت في إعادة كتابة الشفرة.

المفهوم الخاطىء السادس: مادمت لم تنفذ البرنامج فلن تستطيع تقييم جودته.

الحقيقة: كل مرحلة من المشروع البرمجي يجب أن يكون لها تقييم جودة، وعملية الإختبار ليس محددة للمرحلة الأخيرة فقط بل في جميع المراحل.

وكما ذكرنا سابقا فإن اكتشاف الأخطاء مبكرا يوفر الكثير من الوقت والمال.

المفهوم الخاطىء السابع: الشئ الوحيد المطلوب تسليمه في مشروع برمجي هو برنامج يعمل بدون أخطاء.

الحقيقة: البرنامج ليس إلا جزءا من البرمجيات(المنظومة).

وتذكر أن البرمجيات = البرنامج + البيانات + الوثائق.

لاحظ أن التوثيق يشكل الأساس لإعداد منظومة ناجحة وصيانة جيدة.

1.5 مقارنة بين البرمجيات والعتاد: Software vs. Hardware

لدراسة إعداد المنتجات البرمجية Software productsبصورة أعمق نقارنها بعملية إعداد المنتجات المادية أو العتاد Hardware products بالأمس واليوم.

والجدول التالي يبين بعض الفروقات بين مجال الكيان المعنوي (البرمجيات) والكيان المادى

في الحاضر	في الماضي
الآن تشكل البرمجيات البند الأكثر تكلفة (80 ٪ من التكلفة) بينما يشكل العتاد نسبة 20 ٪ فقط من التكلفة.	كان العتاد أكثر تكلفة من البرمجيات، لذلك فقد ركّز الزبائن على العتاد أكثر من البرمجيات.
الآن تستخدم هندسة البرمجيات طرقا وفنيات شبيهة بالمستخدمة في هندسة صناعة العتاد.	كان العتاد يتم تحليله وتصميمه قبل إنتاجه، ولم يكن ذلك الحال مع البرمجيات.
يركز مهندسو البرمجيات الان على ضمان وتأكيد الجودة.	ركز المتخصصون في صناعة العتاد على ضمان الجودة والتحكم في الجودة أكثر من المتخصصين في البرمجيات.
هندسة البرمجيات تنال كل يوم قبولا أكثر كأفضل طريقة لإعداد البرمجيات	كانت البرمجة تعتبر فنا وليس علما (أو هندسة)، لدرجة أن بعض المبرجين كانوا يستعملون طريقة المحاولة والخطأ.
يمكن الآن أن نشتري البرمجيات من المتجر خاصة عند استخدام البرمجة الكائنية (الشيئية) التي تمكّن من إعادة استعمال المكونات البرمجية.	كانت معظم البرمجيات مثل البدلة المفصلة على الزبون، ولم تكن مبنية من مكونات موجودة. ولم يكن هناك دليل لمكونات البرمجيات كما كان للعتاد.

البرمجيات هي عنصر غير مادي في النظام، ولا نستخدم في صناعتها الآلات والمواد الخام، بل نحتاج فقط إلى جهاز حاسوب في عملية الإعداد والتجربة.

وهذه الخاصية للبرمجيات تعطيها مزايا نظرا للأسباب التالية:

- البرمجيات قابلة للتطوير باستمرار (أي بعد الانتهاء من إعدادها يمكن تطويرها أكثر دون أن نبدأ من الصفر).

- تتجه أسعار الكيان المادي (العتاد) نحو الهبوط وبالمقابل فإن أسعار البرمجيات في تزايد، وهذا يعني توفر وظائف أكثر ورواتب أفضل لمعدي البرمجيات
- صناعة البرمجيات تعتمد على التفكير المنطقي والذكاء وليس على رأس المال والمواد الخام، وهذه الحقيقة يجب استغلالها بصورة أفضل وخاصة في الدول النامية لتحسين الأوضاع الاقتصادية.
- إن التعليم السليم والتدريب الموجّه في مجال البرمجيات يمكن أن يكونا استثمارا ذا عوائد مثمرة جدا وخير مثال على ذلك: الهند التى اشتهرت في التعليم والتدريب والاستخدام لتقنية المعلومات.

1.6 أجيال لغات البرمجة Programming Languages Generations

قد تتبادرإلى الذهن الأسئلة التالية:

لماذا نحتاج إلى لغة برمج؟

ولماذا لا يكفي أن نستخدم لغة طبيعية (مثل اللغة الانجليزية أو العربية) لنتخاطب مع الحاسوب؟

وما هي الفروق الأساسية بين لغات البرمجة واللغات الطبيعية؟

الاجابة: إننا نستعمل لغات البرمجة من أجل التخاطب مع الحاسوب لأنه لا يفهم أي لغة من لغات البشر.

ورغم أن البحوث جارية على قدم وساق لجعل اللغة الطبيعية كوسيلة اتصال مع الحاسوب، إلاّ أن هذا الهدف سيحتاج إلى سنوات عديدة لتحقيقه، وحتى ذلك الوقت ستبقى لغات البرمجة هي الوسيلة الفعالة لجعل الحاسوب يعمل ما نريد منه أن يعمل.

والجدول التالي يبين 3 فروقات بين لغات البرمجة واللغات الطبيعية

Natural language لغة طبيعية	Programming language لغة برمجة
المصطلحات ذات مدى واسع	المصطلحات محدودة
قواعد اللغة معقدة	قواعد اللغة صريحة وبسيطة
القواعد والمعاني عادة ما تكون بدون تعريف محدد(مع كثرة المرادفات وتنوع المعنى للمصطلح الواحد).	القواعد والمعاني ذات تعريف محدد .

هذه الفروقات أو (المزايا) الثلاث بين لغات البرمجة على اللغات الطبيعية تجعل من الأولى اللغة السليمة للإتصال مع الحاسوب، ومع ذلك فمن الأفضل أن تكون لغة البرمجة قريبة من اللغة الطبيعية قدر الإمكان وذلك لأنها تيسّر عمل المبرمج كثيرا.

وفي الواقع فإن لغات البرمجة تقسم عادة إلى:

1- لغات الجيل الأول: 1GL (First Generation Languages)

في هذا الجيل تمّ استخدام ما يعرف بلغة الآلة Machine Language . ومشكلة هذه اللغة أن برامجها معقدة وصعبة القراءة لأنها لم تستخدم إلّا رمزين اثنين هما الصفر والواحد .

وثمة مشكلة أخرى وهي أن كل جهاز حاسوب من شركة مصنعة لها شفرتها الخاصة بها ممّا يستوجب المعرفة المسبقة بنوع الجهاز.

2- لغات الجيل الثاني: 2GL (Second Generation Languages)

حدث تحسن طفيف في لغات هذا الجيل فبدلا من استخدام الرمزين الصفر والواحد لتمثيل التعليمات والبيانات في لغة الآلة تمّ استخدام حروف أو مجموعة حروف إنجليزية للتعبير عن التعليمات مثل ADD للتعبير عن الجمع وSUB للتعبير عن الطرح .

واللغة المستخدمة في هذا الجيل هي لغة الاسمبلي Assembly Language

3- لغات الجيل الثالث: 3GL (Third Generation Languages)

تتميز هذه اللغات بأنها أقرب إلى اللغة الطبيعية من لغات الجيل الأول والثاني وأن البرامج المكتوبة بها لا تعتمد على الآلة المستعملة، وهي أبسط في الكتابة والتعديل.

ومن أمثلة لغات الجيل الثالث:

- لغة الفورتران FORTRAN ظهرت سنة 1954 التي تستخدم عادة في التطبيقات العلمية.
- لغة الكوبول COBOL ظهرت سنة 1959 للتطبيقات التجارية.
- لغة باسكال PASCAL ظهرت سنة 1971 لتعليم البرمجة الهيكلية Structured programming.
- لغة C ظهرت سنة 1972 أساسا لغرض برمجة النظم مثل كتابة المترجمات ونظم التشغيل، ولكنها لاقت قبولا في عدة مجالات، ومنها تعليم لغة البرمجة في الجامعات والمعاهد العليا.

4- لغات الجيل الرابع: 4GL (Fourth Generation Languages)

في الجيل السابق كان التركيز على التطبيقات الرقمية التي تعتمد على إجراء العمليات الحسابية مثل إعداد المرتبات والمحاسبة، وكانت البيانات تخزن في ملفات مستقلة وتعالج عادة بلغة برمجة مثل لغة كوبول.

ولكن مع تزايد تعقد التطبيقات وتنوعها في الجيل الرابع استدعى الأمر لغات برمجية تهتم بتخزين واسترجاع البيانات التي تتعامل مع الأسماء والحروف (قواعد البيانات) مع الأخذ في الاعتبار احتياجات الزبون لواجهة رسومية جذابة.

واختصارا لزمن إعداد المنظومات ظهر أيضا مفهوم البرمجة الشيئية التي تستخدم مفهوم إعادة استخدام المكونات أو الأجزاء البرمجية التى تمّ إعدادها في مشاريع سابقة .

لذلك تميزت لغات البرمجة في الجيل الرابع بما يلي:

- برمجة قواعد البيانات Database programming.

- البرمجة الشيئية Object-oriented programming.
- البرمجة المرئية Visual programming.

ومن امثلة هذه اللغات:

- Delphi.
- Visual basic.
- C++.
- Oracle.
- SQL.
- JAVA.

5- لغات الجيل الخامس: 5GL (Fifth Generation Languages)

نظرا للانتشار الواسع للحاسوب واستخدامه بشكل واسع تبنت اليابان مشروعا له علاقة بالذكاء الصناعي Artificial Intelligence لاستخدام اللغات الطبيعية للتعامل والتفاهم مع الحاسوب، إلاّ أن المشروع لم يلاق النجاح ولكن ظهرت وانتشرت في هذا الجيل لغات لبرمجة الروبوت (الإنسان الآلي) ROBOTS في المصانع، وتطورت اللغات التي تتعامل مع هذا النوع من التطبيقات.

كما كان للإنترنت تأثير كبير على نوع البرامج المطلوبة.

لذلك فإن لغات الجيل الخامس تعاملت مع مجالين أساسيين هما:

- الذكاء الاصطناعي Artificial intelligence (AI)
- برمجة الإنترنت Internet Programming

ومن أمثلة هذه اللغات مايلي :

التطبيق الأول (الذكاء الإصطناعي) يشمل لغات مثل LISP و PROLOG.

ويشمل التطبيق الثاني استخدام لغات الانترنت والتي من أمثلتها:

- Java.
- ASP.
- PHP.
- CGI.
- PERL.
- XML.
- HTML.
- VB.net.

الجدول التالي يبين بعض أنواع اللغات وتاريخ ونوع استعمالها

اللغة	السنة	التطبيق
C	1972	برمجة النظم
Delphi	1990	برمجة مرئية و شيئية
Prolog	1972	برمجة الذكاء الإصطناعي
Smalltalk	1980	برمجة شيئية
C++	1985	برمجة شيئية
Eiffel	1992	برمجة شيئية
Java	1995	برمجة شيئية وانترنت

1.7 أنواع البرمجيات: Types of Software

تتنوع البرمجيات على النحو التالي:

1- برمجيات علمية وهندسية : Scientific & Engineering Software

يشار إلى البرمجيات ذات الطابع الحسابي (العددي) أحيانا بالبرمجيات العلمية.

ويجد هذا النوع من البرمجيات تطبيقا واسعا في مجال العلوم التطبيقية مثل علم الفلك Astronomy و تحليل الجهد Stress analysisوالتصميم باستعمال الحاسوب Computer Aided Design والمحاكاة Simulation و التصنيع الآلي Computer-Aided Manufacturing والإحصاء.

2- برمجيات تجارية: **Business Software**

برمجيات تتعامل مع البيانات للحصول على معلومات ضرورية للعمليات التجارية وعمليات اتخاذ القرار.

وهي ذات تطبيقات متعددة مثل :

منظومة المرتبات	Payroll system
منظومة مبيعات	Sales System
منظومة مخازن	Inventory system

3- برمجيات الحاسوب الشخصي **Personal Computer Software**

وهي برمجيات مصممة لتطبيقات الحاسوب الشخصيPC لغرض تحسين إنتاجية الموظفين والمستخدمين على حد سواء.

ومن أمثلة تطبيقاتها :

معالجة الكلمات	Word Processing
الجداول الإلكترونية	Spreadsheets
رسومات الحاسوب	Computer Graphics
إدارة قواعد البيانات	DBMS
الوسائط المتعددة	Multimedia
إدارة الأعمال	Project Management

4- برمجيات الذكاء الاصطناعي **Artificial Intelligence Software (AI)**

برمجيات تجعل الحاسوب يستفيد من (خبرته) في اتخاذ القرار بصورة مشابهة لسلوك الإنسان في التعلم والاستنتاج.

ومن أمثلة تطبيقاتها :

- الأنظمة الخبيرة: Expert Systems مثل منظومة التشخيص الطبي
- التعرف على الصورة والصوتImage and Voice Recognition
- برهنة النظريات Theorem Proving
- الألعاب Games
- الإنسان الآلي Robots

5- البرمجيات المقيمة : **Embedded Software**

برمجيات تقطن في ذاكرة روم ROM للتحكم في الآلات الأخرى مثل الأجهزة المنزلية.

ومن تطبيقاتها:

الفرن (الميكروويفي)	Microwave Oven
المركبات الآلية	Automobile
التحكم في الوقود	Fuel Control
نظام الكابح للسيارات	Braking Systems

6- برمجيات النظم: **System Software**

وهي مجموعة برامج أعدت لخدمة برامج أخرى.

ولها تطبيقات في المجالات التالية:

المترجمات	Compilers

المحررات Editors

خدمات إدارة الملفات File Management Utilities

نظام التشغيل Operating System

7- برمجيات الوقت الحقيقي: Real-time Software

وهي برمجيات تتحكم في الأحداث أثناء وقوعها بحيث يتراوح وقت الاستجابة من 1 ميللي ثانية (أو أقل) إلى دقيقة واحدة.

تتكون برمجيات الوقت الحقيقي من 4 أجزاء:

1. جزء لتجميع البيانات Data gathering component
2. جزء التحليل Analysis component
3. جزء الإخراج Output component
4. جزء تحكم ينسق العمل بين جميع الأجزاء الأخرى لتحقيق الإستجابة الفورية.

ومن أمثلة هذه التطبيقات :

- منظومة التحكم والمراقبة في حركة الطيران.
- منظومة الحجز لخطوط الطيران.

8- برمجيات الإنترنت: Internet Software

وهي برمجيات تتعلق باستخدام الإنترنت ومن أمثلة هذه التطبيقات:

- المصارف بالإنترنت E- Banking
- الادارة الالكترونية E-Government
- التعليم بالإنترنت E- learning
- التجارة الالكترونية E-Trade

1.8 لمحة عن هندسة البرمجيات: Glance at Software Engineering

كان أول استعمال لمصطلح هندسة البرمجيات في مؤتمر عقد في ألمانيا سنة 1968 لمناقشة المشاكل المتزايدة عند إعداد البرمجيات والذي سمي بأزمة البرمجيات، كما تم عقد مؤتمر آخر في إيطاليا سنة 1969 لنفس الغرض، وتم استخدام مصطلح «هندسة البرمجيات» لأول مرة.

وقد تم تقديم المصطلح (هندسة البرمجيات) في أواخر الستينيات للإشارة إلى مشكلة خطيرة في عالم الحاسوب وطرق حلها.

وكما سبق مناقشته في (الفقرة 1.3) فقد كانت هناك العديد من الشكاوي حول المنتجات البرمجية لدرجة وصف المشكلة بالأزمة.

وبعض هذه الشكاوي شملت الآتي:

- إعداد البرمجيات (المنظومات) يتم أحيانا من قبل هواة غير متخصصين
- البرمجيات غالبا ما تحتاج إلى صيانة مستمرة.
- البرمجيات تصميمها كان غير منظم وغير هيكلي مما صعب صيانتها بشكل سليم جعل صيانتها ذات تكلفة باهظة.
- المتطلبات لايتم تنفيذها بالكامل ولا تلبي رغبات الزبون
- تسليم البرمجيات يأتي متأخرا وبتكلفة أكثر مما هو متوقع.
- البرمجيات التي يتم إعدادها ذات أخطاء أثناء تشغيلها ولا يعتمد عليها بالكامل.
- غالبا لا تحقق البرمجيات احتياجات المستخدم كما يجب.
- الأدوات المستخدمة في إعداد البرمجيات ليست ذات كفاءة عالية.

لهذه الأسباب تم تقديم هندسة البرمجياتSoftware Engineering كحل للمشاكل التي تواجه كلا من معدّي البرمجيات ومستخدميها.

أي أن هندسة البرمجيات تهدف لإنتاج برمجيات يمكن الإعتماد عليها وذات كفاءة

عالية وقابلة للصيانة، و يتم ذلك خلال الزمن والتكلفة المحددة وذلك باستعمال أدوات وطرق متينة.

تعريف هندسة البرمجيات

نعطي هنا ثلاثة تعريفات متكافئة:

1. هندسة البرمجيات هي: أسلوب منظم لإنتاج وتشغيل وصيانة البرمجيات
2. هندسة البرمجيات هي: المنهاج الإداري والتقني الذي يؤدي إلى إنتاج وصيانة البرمجيات ذات الجودة في حدود الوقت والتكلفة التقديرية.
3. هندسة البرمجيات هي: دراسة واستعمال طرق وأدوات وأسس وفنيات علم الحاسوب والإدارة والهندسة والاقتصاد وعلم النفس ومهارة التواصل في إعداد وصيانة برمجيات ذات موثوقية(اعتمادية) عالية وقابلة للصيانة والاستعمال في حدود الوقت والتكلفة المتوقعة.

1.9 أهداف وأسس هندسة البرمجيات Objectives & Foundations

لهندسة البرمجيات أهداف يسعى لتحقيقها، وأسس يستند عليها لتحقيقها.

ومن أهم الأهداف التي يسعى إليها علم هندسة البرمجيات الآتي:-

- التحسين في جودة البرمجيات .
- زيادة الإنتاجية في إعداد البرمجيات.
- تحسين طرق الاتصال بين المستخدم ومعد البرنامج.
- التقليل من تكلفة المنتجات البرمجية.

ولكي تتحقق هذه الأهداف فمن المهم توفير درجة عالية من التواصل بين الزبائن والمستخدمين ورئيس وأعضاء إعداد البرمجيات Development team لأن التواصل الجيد - الشفوي والكتابي والشخصي - له أهمية كبيرة في هندسة البرمجيات.

ومن المهم أيضا الدراية بعلم الاقتصاد والإدارة فهو يوفر الأساس لتقدير الموارد والمستلزمات والتحكم في التكلفة.

ولاشك أن المعرفة الأساسية بالعوامل البشرية ضرورية، فهي تؤثر ليس فقط على العمل الجماعي والإبداع، ولكنها تؤثر أيضا على مستويات وقدرات المستخدمين لغرض الحصول على برمجيات يتم قبولها بسهولة من قبل الزبون.

كما يجب أن يكون مهندس البرمجياتSoftware Engineer قادرا على كتابة الوثائق الفنية بوضوح وبدقة.

أسس هندسة البرمجيات:

تعتمد هندسة البرمجيات بصورة كبيرة على معرفة المجالات التالية:

1. علم الحاسوب Computer Science
2. علم الإدارة Management Science
3. مهارات الاتصال Communication Skills
4. الأسلوب الهندسي Engineering Discipline
5. علم الاقتصاد Economics
6. علم النفس Psychology
7. الكتابة الفنية Technical Writing

1.10 أدوات هندسة البرمجيات

يحتاج مهندس البرمجيات إلى أدوات تساعده في عمله تماما كما يحتاجها مهندس العمارة أو أي مهندس آخر.

والطرق والأدوات الآتية يتم استخدامها عادة من قبل مهندس البرمجيات لإعداد البرمجيات:

1. أدوات التحليل لتحديد متطلبات المستخدم.

2. أدوات التصميم لتمثيل الحل.
3. لغات برمجة مناسبة لنوع التطبيق.
4. أدوات ومفاهيم الإدارة لتخطيط ومتابعة المشاريع.
5. إجراءات صيانة البرمجيات.
6. الأدوات الآلية لزيادة جودة البرمجيات وإنتاجية المبرمج والتحكم الإداري للمشاريع البرمجية (مثلا أدوات CASE).
7. إجراءات اعتماد الجودة.
8. طرق لتقدير التكلفة ومدى الاعتماد على المنتوج.
9. منهجيات جديدة لإعداد المنظومات لغرض تقليل وقت وتكلفة المشاريع البرمجية.

1.11 الفرق بين مهندس البرمجيات والمبرمج

ما الفرق بين المبرمج ومهندس البرمجيات؟

الإجابة على هذا السؤال غير واضحة للعديد من الناس، والتعريفات التالية يجب أن تزيل هذا الغموض.

المبرمج : Programmer

المبرمج هو الشخص الذي يهتم بتفاصيل تنفيذ وتعديل الخوارزميات وتراكيب البيانات باستخدام لغة برمجة معينة.

مهندس البرمجيات: Software Engineer

مهندس البرمجيات هو الشخص الذي يهتم بأمور التحليل والتصميم والتنفيذ والإختبار والصيانة والتوثيق والإدارة للمشاريع البرمجية.

من هذه المقارنة نرى أن مهندس البرمجيات له مهام أكثر من المبرمج، فعليه أن ينظر إلى مسألة إعداد البرمجيات من زاوية أوسع.

كما عليه أن يبحث في الجوانب التالية من المشروع :

- ■ التحليل — Analysis
- ■ التصميم — Design
- ■ التشفير أو البرمجة — Coding
- ■ اكتشاف الأخطاء وتصحيحها — Debugging & Testing
- ■ صيانة البرمجيات — SW Maintenance
- ■ التوثيق — Documentation
- ■ إدارة مشاريع برمجية — SW Project Management

> **ملاحظة: يجب أن يكون لمهندس البرمجيات خبرة ومهارة واسعة كمبرمج .**

1.12 تصنيف وظائف هندسة البرمجيات Job Classifications

يوجد الكثير من الوظائف التي نحتاج إليها لتنفيذ دورة حياة إعداد البرمجيات.

نورد بعضا منها في القائمة التالية:

- ■ مهندس التوثيق — Documentation engineer
- ■ مهندس المبيعات — Sales engineer
- ■ مهندس البرمجيات — Software engineer
- ■ محلل النظم — Systems analyst
- ■ مهندس النظم — Systems engineer
- ■ كاتب تقني — Technical writer
- ■ محلل قاعدة البيانات — Database analyst

■ مدير قاعدة البيانات	Database Administrator
■ منسق أمن البيانات	Data security administrator
■ محلل أمن البيانات	Data security analyst
■ مدير تشغيل الحاسوب	Computer operation manager
■ مدير الشبكة المحلية	LAN manager
■ مصمم الرسوم	Graphic designer
■ مدرس علم الحاسوب	Computer science educator
■ إدخال البيانات	Data entry
■ مبرمج	Programmer
■ الدعم الفني	Technical support
■ مهندس الجودة	Quality engineer
■ مدير المشروع	Project manager
■ مبرمج النظم	System programmer
■ فني المكتبة	Librarian
■ مشغل الحاسوب	Computer operator
■ اختصاصي شبكات	Network specialist
■ مصمم موقع وب	Web site designer

ونقوم هنا بوصف بعض هذه الوظائف:

1- مدير المشروع Project Manager

مسئول عن إدارة المشروع البرمجي.

2- محلل النظم Systems Analyst

محلل النظم هو المسؤول عن تجميع وتحليل المتطلبات وكتابة وثيقة مواصفات المتطلبات.

بعد ذلك يستطيع أن يصمم الحل أي يحوّل هذه المواصفات إلى تمثيل للمنظومة باستخدام أدوات مثل المخطط الهيكلي وقاموس البيانات و شبيهة الشفرة .

3- المبرمج Programmer

المبرمج هو الذي يترجم مواصفات التصميم إلى شفرة برمجة باستخدام لغة برمجة مناسبة، كما يجب عليه أن ينفذ هذه الشفرة على الحاسوب للتأكد من صحة عملها.

4- مدير قاعدة البيانات Data Base Administrator (DBA)

مسؤول عن تصميم و إدارة قاعدة البيانات.

5- أخصائيو الرسم بالحاسوب Computer Graphics Specialists

وتتم الإستعانة بهم في تصميم واجهة المستخدم.

6- مدير اتصال البيانات Data Communication Manager

مسؤول على شبكة الإتصالات التي قد تحتاج لها المنظومة.

7- مسؤول مكتبة الحاسوب Librarian

يقوم بحفظ جميع المواد والمعدات المستخدمة والتي يتم الإحتياج إليها في إعداد المنظومة (مثل الأقراص والقوائم والوثائق، الخ...).

الخـلاصة:

هندسة البرمجيات هي عبارة عن تقنية تمزج بين أدوات وطرق وعمليات لغرض الحصول على منتوج برمجي ناجح، وقد حلت هذه الهندسة محل ما كان يعرف سابقا بمعالجة البيانات نظرا لاشتمالها على الجوانب الإدارية والنفسية والإقتصادية المتعلقة بمشاريع البرمجيات.

1.13 أسئلة للمراجعة

1. اذكر فرقا واحدا على الأقل بين المبرمج ومهندس البرمجيات .
2. لماذا من المهم أن يدرس مهندس البرمجيات علم إدارة المشاريع وأن يستخدمه في عملية إعداد البرمجيات ؟
3. ما هو الغرض الرئيسي لهندسة البرمجيات؟
4. بين الصواب من الخطأ :
   - هندسة البرمجيات تعني علم الحاسوب.
   - هندسة البرمجيات هي هندسة النظم.
   - هندسة البرمجيات تنطبق فقط على المشاريع الصغيرة.
   - بدأ العمل في هندسة البرمجيات في أوائل الستينيات.
5. ماذا يعني مصطلح (أزمة البرمجيات)؟ اذكر خمسة مشاكل ذات علاقة بهذه الأزمة.
6. اذكر ثلاثة أسباب تبين أهمية دراسة هندسة البرمجيات.
7. ما هي مراحل إعداد المنظومة ؟ ما هو الهدف من كل مرحلة ؟ وما هي النشاطات الرئيسية في كل مرحلة ؟
8. حدّد 10 أعمال ذات علاقة بهندسة البرمجيات.
9. ما معنى المصطلحات التالية :

Software reuse	Distributed System
Artificial intelligence	AI
Documentation	QA
Software crisis	OOP
Developer	4GL

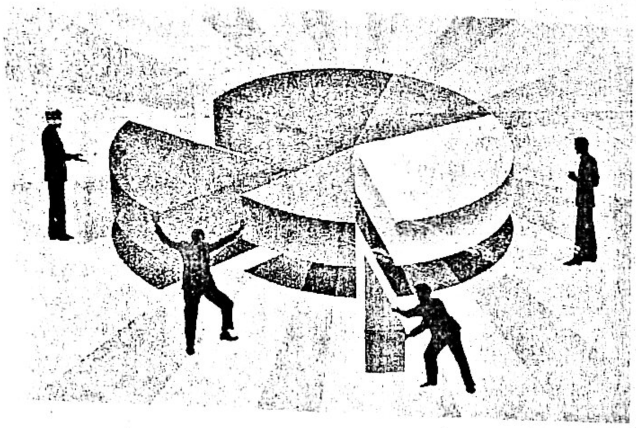
User	Embedded SW
Requirements	ASP
Software quality	DBMS
Natural language	
Multimedia	
Business software	
Custom SW	
Real Time System	

10. ما هي الأجزاء التي تتكون منها البرمجيات؟
11. يقسم تاريخ الحاسوب عادة إلى مجموعة أجيال. ما هي خصائص برمجيات كل جيل؟
12. ماذا يعني مصطلح «أزمة البرمجيات»؟و ما هي المشاكل التي أدت إلى هذه الأزمة؟
13. بين الخطأ والصواب في الجمل التالية مع بيان السبب.
    - إضافة أشخاص للعمل في مشروع متأخر يزيده تأخيرا.
    - يكفي أن نحدد بشكل عام أهداف المشروع لنبدأ في البرمجة.
14. ما المقصود بتعريف المشكلة؟وما الذي يجب أن يشتمل عليه التعريف؟
15. بين بالرسم العلاقة بين تكلفة التغيير، ووقت التغيير.
16. قارن بين عملية إعداد البرمجيات في الأجيال الأولى (الستينيات) والوقت الحاضر، من حيث العمل كفريق، ضمان الجودة، إعادة الاستعمال.

الفصل الثاني

تقدير التكلفة وجدولة المشروع

Cost Estimation and Project Scheduling



2.1 مقدمة

لقد أصبحت البرمجيات في يومنا هذا أغلى تكلفة من العتاد بعكس ماكانت عليه في الستينيات.

لهذا، وعند حصول أي خطأ في تقدير التكلفة قد ينتج عنه خسارة (خصوصا لمعدي المنظومات) ومن ثمّ فشل إعداد المنظومة.

أما تحديد الجدول الزمني فهو مهم جدا، لأنه مرتبط بتقدير التكلفة ويؤثر في نجاح أو فشل المشروع البرمجي، ولهذا تم إعداد طرق لتقدير كل مهمة من مهام المشروع ليتم في النهاية معرفة الزمن الكلي.

لقد سارع العلماء بالاهتمام بهذه القضية الشائكة ومن ثم ايجاد الحلول لها بالقدر المستطاع، حيث استخدموا كل السبل للوصول إلى التقدير التقريبي الواقعي حيث استخدموا متغيرات لها علاقة بالأمور الفنية والإنسانية وحتى البيئة التي يعتقد أنها تؤثر وتساهم في الوصول إلى التقدير الصحيح لتكلفة المنتوج البرمجي وتقدير المدة الزمنية له.

المشروع البرمجي يجب ان يتم بإسلوب إداري سليم.

لمدير المشروع البرمجي العديد من المهام الصعبة، بعض هذه المهام تتعلق بتقدير التكلفة وزمن تنفيذ المنظومة وهي:

- تقدير المدة الزمنية للمشروع.
- تقدير المجهود المتوقع.
- تقدير عدد أعضاء الفريق لأداء هذه المنظومة.
- تقدير للمستلزمات الخاصة للمشروع البرمجي من عتاد وبرمجيات وعنصر بشري وغيرها. ويساعد كثيرا بالطبع معرفة حدود أونطاق المشروع والمخاطر المتوقعة.

2.2 تقدير التكلفة Cost Estimation

يتم تقدير تكلفة المشروع البرمجي عادة لجميع مراحل المشروع بداية بالتقديرات الأولية لمعرفة الجدوى الإقتصادية، وبعد ذلك يتم إعداد تقديرات أكثر تفصيلا عند إعداد خطة المشروع.

وهناك العديد من الطرق والنماذج الشائعة التي تستخدم إمّا الخبرة والإستشارة وإمّا معادلات رياضية (خوارزميات) لحساب التكاليف التقديرية باستخدام متغيرات (بارامترات) مثل عدد أسطر شفرة المنظومة Lines of Code ونقاط الدالة Function points .

ويعتبر استخدام عدد أسطر الشفرة كمعيار لتقدير التكلفة من أشهر القياسات.

أما نقاط الدالة Function Points فتعتمد على العناصر التالية:

- المدخلات.
- المخرجات.
- الإستعلامات.
- الملفات الخارجية.
- الملفات الداخلية.

وهناك مجموعة عوامل أخرى تؤثر أيضا على تقدير تكلفة المشروع نوجز عددا منها:

1. تعقد البرمجية (المنظومة) Complexity of Software

كقاعدة عامة: كلما زاد تعقد المنتوج البرمجي زادت تكلفته.

2. حجم البرمجيات Size of Software

وبصورة عامة فإن البرمجيات ذات الحجم الكبير تكلف أكثر من ذات الحجم الصغير.

3. التقنية المستخدمة Technology

التقنيـة الحديثـة مثـل الأدوات الجديدة في إعـداد البرمجيـات قد تساعـد في تحسين

الإنتاجية، وهذا يمكن أن يؤثر على تخفيض المجهود اللازم في البرمجة وبالتالي تخفيض تكلفة البرمجيات.

4. الاعتمادية Reliability

يمكن التعبير عن الاعتمادية بدلالة الدقة والاكتمال والتناسق في البرنامج المصدر. وبالتالي فإن المنظومات ذوات الاعتمادية العالية تستغرق وقتا أكثر في التحليل والتصميم والتنفيذ والاختبار، وعليه ستكون التكلفة أكثر.

5. أداء المبرمجين Performance of Programmers

يختلف الناس في قدراتهم وذكائهم وحبهم للعمل الجدّي.

وباختيار المبرمجين المجدّين والجيدين يمكن توفير الكثير من تكلفة المشروع.

2.3 إرشادات مهمة عند تقدير التكلفة

تعتبر عملية تقدير تكلفة إعداد المنظومات البرمجية من الأعمال الصعبة خاصة في الحالات التالية:

- عندما لا يوجد بيانات عن مشاريع سابقة مماثلة.
- عندما يتعقد المشروع البرمجي.
- عندما لا تعطي أدوات وطرق تقدير التكلفة نتائج دقيقة.

لتجاوز هذه المشكلة في تقدير التكلفة يمكنك اتباع الإرشادات التالية :

- يجب إعطاء وقت كاف لعمليات تقدير التكلفة (مع التخطيط لها) لأن التقديرات السريعة تولّد تكاليف غير دقيقة .
- استخدم واستفد من بيانات مشاريع سابقة متكاملة وناجحة كلما أمكن ذلك.
- استخدم أسلوب التجزئة Decomposition حيث يتم تقسيم المشروع إلى مجموعة نشاطات ومهمات ثم تجمع تكلفة المهات للوصول الى التكلفة الكلية .

- عند التقدير يجب الخوض في التفاصيل ليصبح التقدير أكثر واقعية.
- خذ في الاعتبار أن عملية التقدير لها صلة بالمجهول ولهذا قد تجلب مخاطر.
- الإستشارة الجماعية تساهم إلى حد كبير في الحصول على التقدير الجيد أفضل من الرأي الإنفرادي .
- استخدم النماذج التجريبية المبنية على الواقعية والاختبار.
- استخدم أكثر من طريقة ونموذج للوصول الى تقدير واقعي ومعقول .

2.4 طرق تقدير التكلفة Cost Estimation Methods

يوجد الكثير من الطرق والنماذج المستخدمة من قبل مديري المشاريع لتقدير تكلفة المشروع، ويمكن استخدام طريقة أو نموذج واحد أو أكثر للمشروع الواحد.

ونبدأ الآن في عرض مجموعة مختارة من الطرق ثم نتبعه بعرض بعض النماذج تباعا.

الطرق الشائعة الإستعمال هي:

1- حكم الخبير Expert Judgement

تعتبر هذه الطريقة الأكثر انتشارا في تقدير التكلفة، حيث يستشار واحد أو أكثر من المستشارين في مجال إعداد البرمجيات في إعطاء تقدير (حكم) عن التكلفة معتمدا على الخبرة Experience والخلفية Background Computer في مجال الحاسوب وحالة السوق Business status.

2- التقدير بالمقارنة Estimation By Analogy

تعتمد هذه الطريقة على تكلفة مشاريع سابقة (تاريخية) من نفس نوع تطبيق المنظومة المقترحة. لهذا تقدر التكلفة بناء على معلومات مأخوذة من المنظومات المنتهية.

3- الفوز بعقد المشروع Project To Win

تعتمد هذه الطريقة على تقدير التكلفة بناء على معلومات عن الميزانية المرصودة من قبل

الزبون لهذا المشروع (المبلغ الذي يكون الزبون مستعدا لإنفاقه على المنظومة) إلى جانب الأخذ بعين الاعتبار ماهي أفضل طريقة لكسب هذا العقد .

ومن عيوب هذه الطريقة أنها لاتهتم بالجوهر(فاعلية ووظائف المنظومة) بل بالشكليات (كسب العقد بأي طريقة) وبالرغم من كون هذا النموذج شاذا وغريبا ولكنه في الواقع يستخدم كثيرا ويعتبر شائعا الى حد ما.

4- الإجماع Group Consensus

تعتمد هذه الطريقة على الرأي الجماعي المقنع والمبني على أساس سليم الذى يصل بنا إلى تقدير واقعي ومعقول.

وتتبع هذه الطريقة الخطوات التالية :

- يقوم منسق هذه الطريقة بإعطاء عدد من المقدّرين(أحصائيين هم باع طويل في تقدير تكلفة المنظومات البرمجية) مستند تعريف للمنظومة ونموذج لتدوين التقدير الابتدائي لتكلفة المشروع.
- يقوم المقدّرون بدراسة دقيقة لمستند تعريف المنظومة، ومن ثمّ إجراء عملية حسابية لتكلفة المشروع وتدوين تقديراتهم الابتدائية في النموذج مع ملاحظة أنه يجب ألاّ يتبادل المقدّرون الأفكار مع بعضهم البعض ولكن بإمكانهم أن يسألوا المنسق عن أي معلومة هم في حاجة إليها.
- يقوم المنسق بجمع نماذج التقديرات وقراءتها ومراجعتها وتلخيصها.
- إذا وجد اختلاف كبير بين التقديرات فيجب تبرير ذلك، وإذا كان التبرير مقنعا يتم رد النماذج مرة أخرى للمقدّرين للحصول على تقدير جديد أفضل.
- تعاد العملية عددا من المرات حتى يتحقق المطلوب مع التأكيد على عدم وجود مناقشة جماعية في أي جلسة عمل.

5 - طريقة من أسفل - إلى - أعلى Bottom-up Method

تعتمد هذه الطريقة على تجزئة المشروع إلى مجموعة مهام أو أجزاءTasks وتقدير كل جزء بذاته، ثم بعد ذلك يتم جمع كل هذه التقديرات المنفردة للحصول على التقدير الكلي للمشروع.

6 - طريقة من أعلى - إلى - أسفل Top –down Method

يتم التقدير بناء على الوظيفة المنطقية وليس على أساس الأجزاء التفصيلية المكوّن لهذا الوظيفة، بمعنى تحديد وظائف المنظومة، وعلاقة الوظائف ببعضها البعض بدون الخوض في التفاصيل.

7 - طريقة قانون باركنسون Parkinson Law

تعتمد هذه الطريقة على مبدأ توزيع مهام المشروع على المدة الزمنية المقررة.

وهذا يعني أن التكلفة التقديرية للمشروع متعلقة ومرتبطة بالموارد المتاحة والمدة الزمنية المحددة أكثر من تعلقها بالأهداف المرجوة من المشروع.

النماذج الشائعة الاستعمال:

تستخدم النماذج كعملية مكملة للطرق السالفة الذكر بحيث يعتمد النموذج على الخبرة السابقة(مثل البيانات التاريخية) وتستخدم معادلات رياضية لحساب التكلفة على شكل دالة بناء على عدد سطور الشفرة (LOC) أونقاط الدالة Function Points كبارامترات لهذه الدالة.

سنستعرض هنا ما يعرف بنماذج البارامترات Parametric Models :

- النموذج الأول ويسمى بنموذج التكلفة Cost Model مثل نموذج كوكومو
- النموذج الثاني ويسمى نموذج القيود Constraint Model مثل نموذج بوتنام سلم.

1 - نموذج كوكومو COCOMO Model

يعطى نموذج كوكومو COCOMO Model تقديرا مباشرا للمجهود المبذول.

ولقد تمّ إعداد هذا النموذج من قبل أحد أقطاب وعلماء هندسة البرمجيات Barry Boahm (في سنة 1981) ويستخدم لتقدير تكاليف المنظومة بطريقة هرمية ويعنى المصطلح نموذج التقدير البنائي COnstructive COst MOdel حيث يحسب هذا النموذج التكاليف (أو المجهود) كدالة لحجم المنظومة Program Size معبرا عنها بعدد سطور شفرة المنظومة Lines of Code (LOC)

ويتكون نموذج COCOMO من عدة نماذج فرعية وهي:

- النموذج الأساسي Basic Model ويتم إعداده في بداية المشروع (أي مرحلة التخطيط).
- النموذج المتوسطIntermediate Model يتم إعداده عند تجهيز وثيقة المتطلبات (أي مرحلة التحليل).
- النموذج المتطور Advanced Model (الأخير): ويتم تجهيزه في نهاية مرحلة التصميم.

2 - نموذج بوتنام Putnam

أعدّ الباحث Putnam نموذج القيودات Constraint model والذي سماه SLIM وقد تم إعداده ليناسب المشاريع البرمجية(المنظومات) الكبيرة أي التي تزيد عن 50000 سطر من شفرة المصدر.

فهذا النموذج يعتمد على مجموعة خصائص للوصول إلى تقدير التكلفة.

وهذه الخصائص هي:

- المدخلات Inputs.
- المخرجات Output.
- ملفات البيانات Data files.
- الاستعلامات Inquires.
- البينيات الخارجية External Interfaces (واجهة المستخدم من شاشات وتقارير).

ويستخدم مخطط المشروع قيما تفاؤلية Optimistic Values وقيم محتملة Most Likely values وقيما تشاؤمية Pissimistic values لكل خاصية من الخصائص السابقة مستعينا بالخبرة السابقة من مشاريع شبيهة.

ومن هده القيم يمكن حساب القيم المتوقعة (التقديرية)Ev=Estimated value حسب المعادلة الاتية:

$$Ev = (Ov + Pv + 4*MLv)/6$$

حيث:

EV	القيمة المتوقعه	Expected value
Ov	القيمة التفاؤلية	Optimistiv Value
Pv	القيمة التشاؤمية	Pissimistic Value
Mlv	القيمة الأكثر ترجيحا	Most likely value

الخلاصة لموضوع تقدير التكلفة باستخدام النماذج:

إن أغلب النماذج المستخدمة تتطلب استعمال عدد أسطر الشفرة LOC لقياس الحجم SIZE وهذا الحجم يتم استخدامه كمعيار لتقدير التكلفة بالرغم من الصعوبة في تقديره، خصوصا في المراحل الأولى من المشروع ونعنى به مرحلتي التخطيط والتحليل.

أمّا معيار نقاط الوظيفة Function points فيمكن استخدامه في المراحل المتقدمة من المشروع أي مرحلتي التصميم والتنفيد.

أدوات التقدير باستخدام الحاسوب Automated Estimation Tools

وهي رزمة برمجيات جاهزةSoftware package تستخدم لمساعدة مدير المشروع في الحصول على تقديرات معقولة للتكاليف والمدة الزمنية وقيمة المجهود المبذول .

وتستخدم طريقة أو مجموعة من الطرق والنماذج السابقة مع استخدام واجهة مستخدم مناسبة لعرض النتائج.

وفي استخدام أدوات التقدير يلاحظ ما يأتي :

- للبيانات التاريخية الخاصة بمشاريع سابقة أهمية قصوى فهي تعتبر العصب المهم في العملية التقديرية وبدونها يصبح أساس التقديرهشا وغير متماسك شأنه شأن البناء المهتز بدون قواعد سليمة .
- إن تقدير التكلفة يعتمد أكثر مايعتمد على الخبرة .
- في المراحل الأولى من المشروع يصعب التخمين والتقدير الأمثل لأن عدد سطور الشفرة لانحصل عليه إلا في نهاية التصميم او في مرحلة التنفيذ.
- يعتبر نموذج كوكومو الأنسب والأكثر شيوعا بين متخصصي البرمجيات فهو يأخذ في الاعتبار الخصائص المتعلقة بالمشروع والمنظومة والعتاد والعنصر البشري على حد سواء عند تحديد التقدير الخاص بالتكلفة ويمكن أن يساهم أيضا في تقدير المدة الزمنية.
- في حالة وجود البيانات التاريخية التي تضاف لنموذج كوكومو يكون النموذج أكثر دقة.

2.5 جدولة المشروع البرمجي Software Project Scheduling

الجدولة للمشاريع سواء كانت برمجيات او مشاريع إنشاءات او مشاريع صناعية تعتبر إلى حد كبير متشابهة وتستخدم نفس الأدوات، لهذا يمكن استخدام أدوات الجدولة مع إجراء بعض التعديلات لتواكب المشروعات الخاصة بالبرمجيات.

وتعتبر جدولة المشاريع من أصعب المهام التي تجابه مدير المشروع أو المتخصص في إعداد خطة المشروع، والطريقة الأساسية المستخدمة هي الإستفادة من جداول المشاريع السابقة .

الا ان هذا لايكفي نظرا لاختلاف البرمجة من مشروع لاخر او اختلاف المنهجيات والأدوات المستخدمة في أغلب المشاريع البرمجية.

ويتم حساب الزمن الكلي لانهاء المنظومة بتجزئة العمل الكلي الى مجموعة مهام جزئية مع الأخذ في الاعتبار أن بعض المهام يتم بالتوازي والبعض الاخر يتم بالتسلسل لأنه يعتمد على بعضه الأخر.

هذا وإن كل مرحلة لابد أن تتسم ببعض المشاكل، وهذه المشاكل تسبب في تأخير تسل
شروع ومن هذه المشاكل على سبيل المثال:

- مرض غياب أو تاخير بعض أعضاء المنظومة، خصوصا الأعضاء الفاعلين.
- تعطل بعض العتاد مثلًا بسبب خلل في التيار الكهربائي.
- عدم وصول برمجيات أوتأخير التدريب، خصوصا عندما تستخدم تقنية متطورة .
- حداثة واختلاف محتوى المنظومة عن أي منظومة قد تمّ إعدادها مسبقا.

والطريقة المتبعة هي التقدير على أساس أن كل شيء يسير على مايرام، ثم إضافة نسبة مئوية (ولتكن 10 ٪) بسبب المشاكل التي قد تعترض العمل.

ولتقدير الزمن الكلي نبدأ بتقدير حجم المنظومة، ثمّ نقسم على إنتاجية المبرمجين المتوقعة لنحصل في نهاية المطاف على عدد الأشهر لكل مبرمج.

وفي هذا الباب سنتعامل بتفصيل أكثر مع عملية تقدير الوقت والتكلفة للمشروع البرمجي، إلى جانب دراسة الأدوات المساعدة باستخدام الحاسوب في هذه العملية.

إن الجدولة وتقدير الوقت اللازم للمشروع يعتبر من أصعب الأجزاء في التخطيط حيث يعتمد التقدير الزمني لأي مشروع على الفترة الزمنية التي يستغرقها انجاز كل عمل جزئي من هذا المشروع.

وعندما نقدر الزمن يجب أن نستعمل أيام العمل كوحدة قياس للزمن مع ملاحظة أن ساعات الدوام قد تختلف من بلد إلى اخر وفي هذه الحالة تستخدم الساعة كوحدة قياس للزمن.

إرشادات بخصوص تقدير الوقت:

- يجب التعرف على المهام التي يمكن أن تنجز على التوازي (أي التي لا تعتمد على بعضها البعض).
- استعمل أقل قوى عاملة ممكنة.
- خذ في الاعتبار أن بعض الناس قد يتركون العمل أو يتعرضون للمرض.

- ليكن في معلومك أن مواصفات المتطلبات والتصميم تتطلب عادة ضعف الوقت اللازم لكتابة شفرة البرنامج.
- أدوات الجدولة يجب أن تبين الآتي :
  - مهام المشروع Project tasks.
  - العلاقة بين المهام.
  - المدة الزمنية لكل مهمة.
- المخططات إمّا أن يتم رسمها يدويا، أو بأدوات خاصة مثل برنامج «المدير - MS Manager» الذي أعدته شركة مايكروسوفت.

مثال: إذا كان الجزء البرمجي يستغرق يوما واحدا لإنهائه، وكان لدينا 30 جزءا برمجيا في المشروع، فإننا نقدر الزمن لكتابة كل الأجزاء بمدة 30 يوما ثم يضاف إلى ذلك 3 أيام (أي 10٪) للأحداث غير المتوقعة (مرض، إجازة، ..) فيصبح التقدير هو 33 يوما.

2.6 أدوات الجدولة Tools of Scheduling

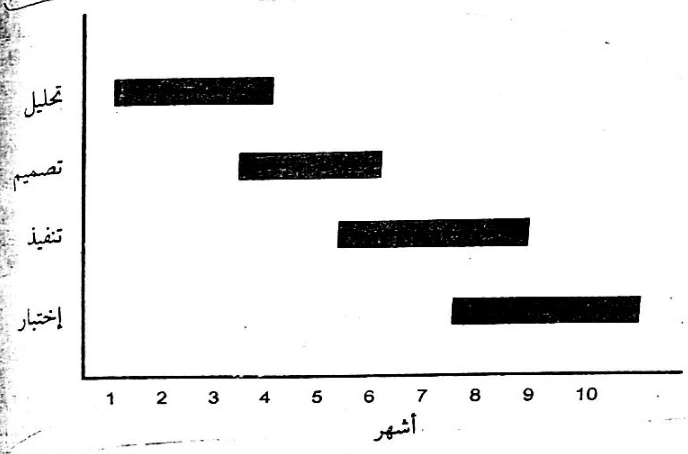
هذه الأدوات تفترض أن المشروع يتكون من مجموعة من النشاطات أو المهام Activities or tasks المعتمدة على بعضها البعض، فمنها مهام لا يمكن أن تبدأ قبل أن تنتهي مهام أخرى، ولكن بعض المهام (أوالنشاطات) يمكن أن تنجز بالتوازي مع مهام أخرى.

وفي عملية الجدولة يستخدم مديرو المشاريع أدوات مثل: مخططات غانت Gantt Chart وجدول الخط الزمني Time-line table، وطريقة CPM وطريقة PERT وهذه الأدوات تستخدم في كل من التخطيط ومتابعة المشاريع.

2.6.1 مخطط غانت Gantt Chart

يعتبر مخطط غانت Gantt Chart أداة تخطيط رسومية تستخدم المحور الأفقي والمحور الرأسي في جدولة المشروع على شكل قضبان أفقية حيث يستخدم المحور الأفقي لزمن التنفيذ والمحور الرأسي لأسماء النشاطات أو المهام (انظر الشكل 2.6.1).

ويمكن أن يبين هذا المخطط أيضا الأشخاص المسؤولين عن هذه المهام.



الشكل (2.6.1) مخطط غانت

2.6.2 طريقة المسار الحرج Critical Path Method (CPM)

عبارة عن أداة تخطيط على شكل شبكة تبين المهام Tasks والمدة الزمنية المقدرة لكل مهمة Estimated time.

وتعتمد هذه الطريقة على استخدام مايسمى بالمسار الحرج Critical path وهو أطول مسار من البداية الى النهاية . وهذا المسار يعطينا أقصر مدة زمنية ممكنة لانهاء المشروع.

لقد تم اعداد طريقة المسار الحرج في الستينيات وكانت تستخدم غالبا في أعمال البناء، ثم أصبحت وسيلة مناسبة في العديد من التطبيقات بما في ذلك تطبيقات البرمجة.

وتتلخص هذه الطريقة في تحديد كل المسارات الممكنة من البداية للنهاية وجمع المدد الزمنية لكل منها. ثم نحدد أطول هذه المسارات والمسمى بالمسار الحرج.

مثال:

يبين الشكل (2.6.2) شبكة CPM حيث يوجد لدينا 8 مهام هي:

T1، T2، T3، T4، T5، T6، T7، T8

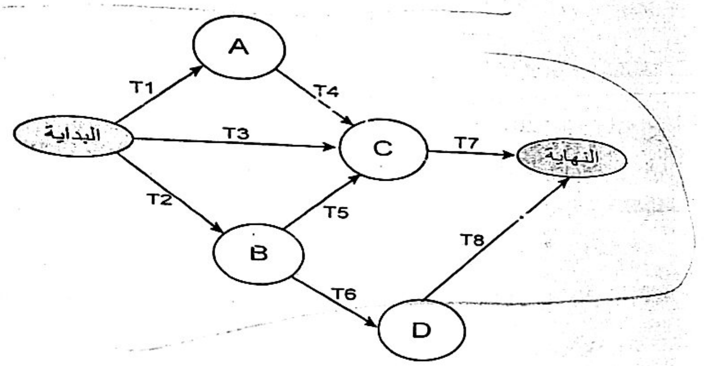
فمثلا قد تكون المهمة:

T1 المقابلة الشخصية

T2 الملاحظة

T3 العرض التجريبي

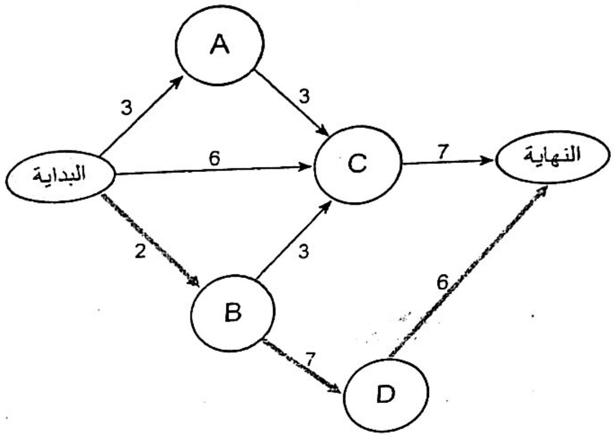
وهذه المهام يمكن أن تبدأ في نفس الوقت.



الشكل (2.6.2) شبكة CPM

لاحظ أن الأحداث A وB وC وD تبين إمّا بداية أو نهاية الأحداث وتسمى حدث Mile-stone/ Event لذلك يمكننا كتابة تاريخ نهاية المهمة فوق الحدث.

ونضع المدة الزمنية الخاصة لكل مهمة، ولتكن بالأيام، كما في الشكل التالي (2.6.3) :



الشكل (2.6.3) المدد الزمنية والمسار الحرج

يبين الشكل (2.6.3) المدة الزمنية المتوقعة (بوحدات الزمن) للمهام المبينة بالشبكة بالشكل (2.6.2) وبتتبع المخطط نلاحظ أن المهمتين T1 وT2 يمكن إنجازهما في نفس الوقت.

وللوصول إلى الحدث C لدينا 3 مسارات:

$$T1 + T4 = 3 + 3 = 6$$

$$T3 = 6$$

$$T2 + T5 = 2 + 3 = 5$$

وهذا يعني أننا لا نستطيع أن نبدأ المهمة T7 قبل مرور 6 وحدات زمنية من بداية المشروع.

وبصورة عامة، ولحساب أقصر وقت يلزم لإنجاز جميع مهام المشروع، ننتقي أطول مسار زمني من بين جميع المسارات الممكنة في الشبكة للوصول إلى نقطة النهاية.

على سبيل المثال نجد في الشكل (2.6.2) أن للوصول إلى نقطة النهاية يمكننا اتباع أحد المسارات التالية:

$$T1 + T4 + T7 = 3+3+7 = 13$$

$$T3 + T7 = 6+7 = 13$$

$$T2 + T6 + T8 = 2+7+6 = 15$$

$$T2 + T5 + T7 = 2+3+7 = 12$$

لذلك فإن أطول مسار من بين المسارات المبينة أعلاه هو المسار الثالث الذي يحتاج إلى 15 وحدة زمنية (15 يوما مثلا).

وهذا يعني أن المسار الحرج كما هو موضح سالفا في الشكل (2.6.3) هو T2+T6+T8 .

إذا افترضنا أن المشروع قد بدأ يوم 1/ 1/ 2008 فتكون أقصر مدة زمنية ممكنة هي 15 يوما أي ينتهي المشروع يوم 16/ 1/ 2008.

2.6.4 أداة برت PERT

تعتبر أداة طريقة التقييم والمراجعة للمشروع Project Evaluation & Review Technique Pert من الأدوات التي تمكننا من الحصول على زمن تقديري لإنهاء المشروع بدقة اكثر من المخطط السابق CPM أي أنه تحسين للطريقة السابقة.

وبعبارة أدق تحسين الزمن التقديري لكل مهمة Ev، حيث لكل مهمة (نشاط) نستخدم ثلاثة تقديرات للزمن اللازم لها:

تقدير متفائل، وتقدير متشائم، وتقدير واقعي (أي الأكثر احتمالا).

ويحسب الزمن المتوقع لكل مهمة (نشاط) كما يلي:

$$\frac{OT + PT + 4\,ML}{6} = ET$$

حيث:

ET = expected time الزمن المتوقع

OT = optimistic estimate الزمن المتفائل

PT = pissimistic estimate   التقدير المتشائم

ML = most likely time   الزمن الأكثر احتمالا

مثال:

افترض أن لدينا المهام ((Tasks التالية لإنهاء مشروع:

Task Number	OT	PT	ML	ET
Task 1	3	8	7	-
Task 2	4	8	6	-
Task 3	5	9	7	-
Task 4	3	5	4	-
Task 5	4	7	5	-

ما هي المدة المتوقعة لإنهاء المشروع؟

الحل:

نستخدم هنا الصيغة أو المعادلة السابقة لحساب الزمن المتوقع لكل مهمة:

$$\frac{OT + PT + 4\ ML}{6} = ET$$

فنحصل على المدة المتوقعة كما هي مبينة في الجدول التالي:

Task Number	OT	PT	ML	ET
Task 1	3	8	7	6.5
Task 2	4	8	6	6.0
Task 3	5	9	7	7.0
Task 4	3	5	4	5.3
Task 5	4	7	5	5.2

بعد حساب الزمن المتوقع لكل مهمة، نستطيع أن نستخدم طريقة CPM لحساب أقصر وقت ممكن لإنهاء المشروع.

نستطيع أيضا أن نضيف عمودين لهذا الجدول الأول للتكلفة المتوقعة والثاني للتكلفة الفعلية.

2.7 أسئلة للمراجعة

1. ما هي العوامل التي تؤثر على تقدير التكاليف لمشروع برمجي؟
2. لماذا توجد صعوبة في تقدير تكاليف المشاريع البرمجية؟
3. بيّن بعض الطرق المستخدمة في تقدير تكاليف المشاريع البرمجية.
4. اذكر بعض التوجيهات التي تساعد في تقدير الوقت اللازم لتحقيق مشروع برمجي.
5. إذا كان لديك البيانات التالية لتقدير الزمن (بالأشهر) اللازم لمشروع برمجي:

a. Analysis	التحليل	3 months
b. Design	التصميم	2 months
c. Coding	التشفير	3 months
d. Testing	الاختبار	4 months

ارسم مخطط CPM للمشروع بافتراض أن كل مرحلة تبدأ بمجرد انتهاء المرحلة السابقة لها.

6. افترض أن مشروع بناء يمر بالمراحل التالية:

■ Initial Design	التصميم المبدئي:	3 months
■ Drawings	الخريطة:	3 months
■ Skelton	الهيكل:	4 months
■ Utilities	الخدمات:	5 months
■ Finishings	الإنهاء:	4 months

ارسم مخطط CPM للمشروع بافتراض أن جميع المراحل تتم تتابعيا أي مرحلة بعد أخرى ولكن المرحلة الأخيرة يمكن أن تبدأ مبكرا لمدة شهر واحد.

7. ارسم شبكة CPM للبرنامج الدراسي في كلية.

افترض أن مدة المقرر الواحد هي فصل دراسي :

Course المقرر	Prerequisite التمهيدي
CS1	NONE
CS2	CS1
CS3	CS1
CS4	CS2
CS5	CS2
CS6	CS3
CS7	CS5
CS8	CS4+CS7

8. أوجد المسار الحرج في المسألة (7). ما هي أقصر مدة زمنية يحتاجها الطالب لإنهاء البرنامج؟

9. تستخدم CPM لحساب أقصر مدة زمنية لإنهاء مشروع أكثر من الطرق الأخرى (مثلا طريقة غانت). لماذا؟

10. من البيانات التالية للفترات الزمنية المتوقعة لمهام المشروع :

	OT	ML	PT
Task1	4	5	6
Task2	3	5	7
Task3	2	4	7
Task4	5	6	8

احسب الوقت المتوقع لكل مهمة.

11. ارسم مخطط غانت لمشروع برمجي نموذجي افترض أن المشروع يتكون من:

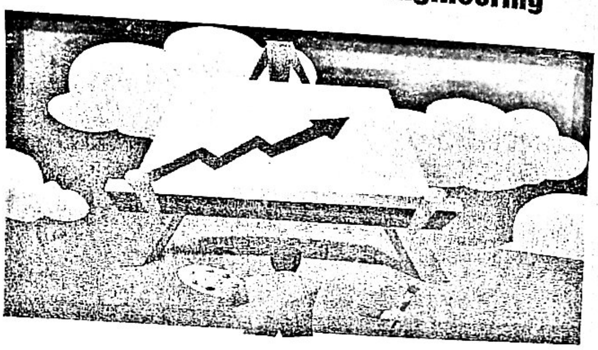
- Project plan	خطة المشروع	wk 1
- Data gathering	تجميع البيانات	wk 1
- Data analysis	تحليل البيانات	wk 2
- Requirements	تحديد المتطلبات	wk 1

- Initial design	تصميم مبدئي	wk 1
- Detailed design	تصميم مفصل	wk 3
- Coding	التشفير	wk 6
- Testing	الاختبار	wk 3
- Documentation	توثيق	wks 4

الفصل الثالث

هندسة المتطلبات البرمجية

Software Requirement Engineering



3.1 مقدمة

إن أحد الأسباب الأساسية لفشل معظم المنظومات البرمجية هو وجود مشاكل في تحديد المتطلباتRequirements، ويمكن أن نقول بأن تقريبًا 85 ٪ من الأخطاء في المنظومات مردّه ومرجعه عدم وضوح المتطلبات وبالتالي المساعدة على عرقلة المراحل التي تلي مرحلة التحليل، وفي النهاية الحصول على منظومات غير ذات جودة ولا تلبي احتياجات المستخدم بالكامل.

لقد كان الاهتمام في:

- الستينيات بالبرمجة.
- السبعينيات بالتصميم.
- الثمانينيات والتسعينيات (وإلى حد الآن) بالمتطلبات والمواصفات.

وتعتبر المرحلة المتعلقة بجمع المتطلباتRequirement Gathering (مرحلة التحليل) من أهم مراحل دورة حياة إعداد المنظومة لأن جمع المتطلبات إذا لم يتم بالصورة الصحيحة والكاملة، فإن كل المراحل التي تليها ستكون بالطبع غير كاملة.

ومهما يكن مستوى التصميم جيدا فلن تنجح المنظومة إلاّ بجودة المتطلبات أولا.

وهناك مفهوم خاطىء عند بعض الناس مردّه أن المتطلبات موجودة في رأس المستخدم أو الزبون. والحقيقة التي تمّ اكتشافها هي أن جمع المتطلبات عمل جماعي يتم أداؤه من قبل ما يسمى Stakeholders وهم الزبون والمستخدم ومعد المنظومة.

لهذا بدأ العلماء والمتخصصون والهيئات العلمية ذات العلاقة بهذا المجال في التفكير جديا بالتعامل مع هذه المسألة وإيجاد الحلول لها ومن ثمّ ظهر مصطلح هندسة المتطلبات البرمجية Requirement Engineering يطفو على السطح في بداية التسعينيات وهو نفس النمط الذي

ظهر في نهاية الستينيات عندما ظهر مصطلح هندسة البرمجيات كحل لأزمة البرمجيات Software crisis التي لم تحل بالكامل حتى الآن.

فهندسة المتطلبات البرمجية ستساهم مع غيرها من الطرق لحل هذه المشكلة التي شعرت بها الشركات المتخصصة لإعداد المنظومات وأدركت ان العواقب ستكون وخيمة وقوية وتؤثر على الإعداد والتكلفة والصيانة.

وقبل أن نبدأ في شرح محتويات هذا الفصل نورد بعض التعريفات المهمة للأشخاص الذين لهم علاقة بهندسة المتطلبات وهم:

- المستخدم النهائي End-user: وهو الشخص الذي سيستخدم المنظومة لأداء الوظائف المنوطة به.
- مهندس البرمجيات Software Engineer: الشخص الذي له الخبرة والمعرفة بالتحليل والتصميم والبرمجة والاختبار وإعداد النماذج التجريبية وغيرها.
- مهندس المتطلبات Requirement Engineer: الشخص المسئول على التعرف على المتطلبات وتوثيقها.

نستعرض في هذا الفصل معنى ومشاكل المتطلبات ثم نعرّج على خمس نشاطات رئيسة خاصة بالوصول إلى متطلبات صحيحة وكاملة وواضحة، وهذه النشاطات تشتمل على:

- استنباط المتطلبات Requirement Elicitation
- تحليل المتطلبات Requirement Analysis
- توصيف المتطلبات Requirement Specification
- اعتماد المتطلبات Requirement Validation
- إدارة المتطلبات Requirement Management

ونستعرض بشيء من التفصيل إدارة المكونات البرجية، فهي الإدارة التي تتعلق بمتابعة التغيير.

عرف مصطلح المتطلبات وماهي أنواعها

3.2 المتطلبات ومشاكلها Requirements And its problems

المتطلبات ما هي إلا ملخص لمجموعة طلبات يرغبها الزبون في شكل وظائف وقدرات وخصائص ومعايير جودة خاصة بالمنظومة المعلوماتية المطلوبة وذلك لجلب منفعة وقيمة للمستخدم.

أو بتعبير آخر هي قدرات برمجية مطلوبة في المنظومة لحل مشكلة المستخدم.

أنواع المتطلبات البرمجية:

يمكن تصنيف المتطلبات البرمجية حسب الآتي:

1 - متطلبات وظيفية Functional Requirements

المتطلبات الوظيفية هي الأفعال (الوظائف) المرغوب أداؤها بواسطة المنظومة البرجية. أو هي المعاملات الرئيسة التي تحدث في المنظومة من حيث قراءة ومعالجة البيانات والحصول على النتائج في صورة مخرجات.

2 - متطلبات الأداء Performance Requirements

وتسمى متطلبات زمن الاستجابة مثل: كم عدد المعاملات التي يتم تنفيذها في فترة زمنية محددة أو أن زمن الاستفسار عن صنف معين لا يتعدى ثلاث ثوان أو نوع وكمية البيانات التى يتم التعامل معها ومعالجتها. فمثلا:

قد تكون إحدى المتطلبات مايلي:

95 ٪ من المعاملات يجب أداؤها في زمن أقل من الثانية الواحدة.

3 - متطلبات البيانات Data Requirement

البيانات المطلوبة في المنظومة الجديدة.

4 - متطلبات التشغيل Operational Requirements

يمكن أن تشتمل على مؤهلات وقدرات المستخدم ومعايير واجهة المستخدم المطلوبة.

5 - متطلبات القبول Acceptance Requirements

وتتضمن خصائص الجودة المطلوبة في المنظومة البرمجية مثل الاعتمادية وسهولة الصيانة والحماية والكفاءة وغيرها وستتعرض لهذه الخصائص بالتفصيل في الفصل الخاص بجودة البرمجيات، وأيضا أنواع الاختبارات والتوثيق والتدريب والأشياء التي ستسلم للزبون مع المنظومة.

المتطلبات: المشاكل والأسباب

تتلخص أهم المشاكل والأسباب التي تتعلق بالمتطلبات في الآتي :

- الزبون أو المستخدم لا يعرف بالضبط ماذا يريد وماهي احتياجاته الفعلية.
- التحليل في أغلب الأحيان ناقص وغير كامل.
- التغيير الكبير في تقنية الحاسوب (البرمجيات والعتاد) وعالم التجارة والأعمال زاد من الضغط على تقليص مدة إعداد المنظومة.
- معد البرمجيات يجد صعوبة في ترجمة أو تحويل المتطلبات إلى برامج وقواعد بيانات.
- التغيير في المتطلبات من جانب الزبون متكرر وهو في بعض الأحيان غير مبرر.
- الميزانية والجدول الزمني غير محمول وغير مقبول.
- التغيير في اللوائح والقوانين زاد من مشكلة المتطلبات.
- عدم تعريف نطاق العمل المراد أداؤه بصورة صحيحة وكاملة.
- التعرف على كامل المتطلبات وتوثيقها صعب للغاية.
- عدم قدرة المستخدم على فهم أهمية وغرض مواصفات المتطلبات.
- عدم القدرة على فهم منهجية صحيحة عند إعداد المنظومات.
- الاعتقاد السائد بين المدراء بأن مرحلتي البرمجة والاختبار هما المجهود الحقيقي والمهم في أي مشروع برمجي دون مراعاة أهمية المراحل الأخرى.
- نقص في مدراء مشاريع البرمجيات المهرة وذوي الخبرة.

وفي الحقيقة أن أغلب الأخطاء في المتطلبات البرمجية ناتج عن فشل معدي البرمجيات في الوصول إلى الأهداف الآتية:

- الفهم بالكامل للوظائف المطلوبة من المنظومة.
- التواصل بهذه المتطلبات مع الأشخاص ذوي العلاقة بالمنظومة Stakeholders مثل الزبون والمستخدم ومحلل النظم وغيرهم.
- إيجاد طريقة ناجحة للتأكد والتحقق من أن المنظومة ستلبي المتطلبات.

3.3 هندسة المتطلبات البرمجية وأهدافها

هندسة المتطلبات البرمجية Software Requirement Engineering هي كل النشاطات المستخدمة للتعرف على المتطلبات ثم تحليل هذه المتطلبات للوصول إلى متطلبات إضافية ومن ثم توثيق واعتماد هذه المتطلبات لتلبي احتياجات المستخدم.

أما الأهداف التي ترمي إليها فهي :

1. فهم أساسيات هندسة المتطلبات
2. التعرف على طرق استنباط المتطلبات
3. تحليل المتطلبات البرمجية
4. تكوين وكتابة مواصفات المتطلبات
5. تقييم واعتماد المتطلبات البرمجية
6. إدارة أي تغييرات تحصل لهذه المتطلبات

ولنبدأ بوظائف هندسة المتطلبات البرمجية :

3.4 استنباط المتطلبات Elicitation of Requirements

يعتبر هذا النشاط، وهو جمع واستنباط المتطلبات من المستخدم والزبون الوظيفة الأولى

في مرحلة التحليل ويتم فيه الاستفادة من كل الذين لهم علاقة بالمنظومةStakeholders المراد إعدادها من أجل اكتشاف واستنباط وفهم احتياجات المستخدم User needs.

إن أصعب مهمة في عملية جمع المتطلبات ليس توثيق ما يريده الزبون، ولكن محاولة مساعدة الزبون لفهم ما يحتاجه.

ويتم في هذا النشاط استخدام مجموعة طرق وأدوات لهذا الغرض منها:

3.4.1 المقابلة الشخصية Interview

تعتبر المقابلة الشخصية إحدى أهم طرق جمع المتطلبات الخاصة بالمنظومات المعلوماتية، وفي بداية المشروع يقضي محلل النظم وقتا كافيا لعمل مقابلات شخصية مع الزبون والمستخدم من أجل فهم طبيعة العمل والبيانات والمعلومات المتداولة في بيئة الزبون وأيضا القوانين واللوائح المستخدمة لتسيير دفة العمل.

وهناك عدة إرشادات يجب اتباعها من قبل المحلل حتى تكون المقابلة ناجحة وتؤدي الغرض المطلوب منها:

التجهيز الجيد للمقابلة من حيث التوقيت، ومعرفة غرض المقابلة من قبل الزبون والمستخدم، مع الاستماع الجيد من قبل المحلل وتنوع الأشخاص المراد التعامل معهم بحيث تجرى المقابلات مع المديرين لتحديد نطاق العمل والأهداف المرجوة.

أمّا الموظفون والمستخدمون وبحكم معرفتهم بالعمل الميداني فتجرى معهم مقابلات تفصيلية حيث يتم توجيه أسئلة متنوعة ذات الإجابة القصيرة والعفوية لمعرفة دور كل منهم والمشاكل التي تعترضهم.

3.4.2 الاستبيان Questionnaire

بالرغم من أهمية المقابلة الشخصية كطريقة من الطرق الرئيسية في الحصول على المتطلبات إلاّ أنها مكلفة ويتم إجراؤها على عدد صغير ومحدود، بمعنى أنه لن تكون الصورة واضحة على كل ما يجري.

لهذا فإن طريقة الاستبيان Questionnaire هي النقيض حيث تجرى على عدد هائل من الناس في زمن قليل. وعادة يتم تجهيز الاستبيان على ورق، الاّ أنه يمكن إجراؤه على الهاتف أو الانترنت.

ويحتوي الاستبيان عادة على مجموعة من الأسئلة القصيرة والطويلة أو الأسئلة ذات الإجابة المحددة من بين مجموعة إجاباتMultiple Choice .وتجري الاستبيانات وتسمى أيضا المسح Survey للوصول إلى هدف معين. وعلى سبيل المثال يمكن استخدامها لمعرفة رأي المستخدمين في عمل النظام(أو المنظومة) الحالي.

3.4.3 الملاحظة Observation

في الطريقتين السالفتين كانت المعلومات تأتي مباشرة من المستخدمين والزبائن، وبالتأكيد قد تمّ الحصول على الكثير من المعلومات.

ولكن السؤال هو ما مدى مصداقية هذه المعلومات ؟ ومدى انطباقها على الواقع الفعلي. إن الحقيقة تقول أن دقة وكمال المعلومات لن تكون مستوفاة حتى لو حاولوا قدر جهدهم إعطاءها حقها، وهذه طبيعة البشر: عدم الدقة والنقصان.

لهذا وللتحقق من المعلومات التي تمّ جمعها من المقابلة والاستبيان يتحتم على محلل النظم أن يشاهد الموظفين أثناء أدائهم لأعمالهم ويراقب العمل الفعلي على أرض الواقع ويتقصى الحقائق ميدانيا.

فالملاحظة Observation تعتبر طريقة مباشرة لتفقد ومعرفة النظام تحت الدراسة وتساعد على الوصول للمعلومة بشفافية وبموضوعية كما هي وليس كما يعتقدها الزبون والمستخدم. فهذه الطريقة تساعد محلل النظم على معرفة (أين ومتى وكم ومن)

- من الذي قام بأداء العمل؟
- ماذا أدى من عمل؟
- أين تمّ أداء العمل؟
- لماذا تمّ أداء العمل؟

- ■ ما المدة الزمنية التي استغرقت لأداء العمل؟
- ■ كيف تمّ أداء العمل؟

وبالتأكيد تتيح إجابات هذه الأسئلة دائمًا لمحلل النظم إزالة الالتباس أو الغموض حول النظام تحت الدراسة.

ولكن من عيوبها أن عملها ليس دائم ومستمرًا، لهذا قد يلتجئ المستخدم للتصنّع والعمل المثالي لحظة ملاحظة المحلل وبالتالي تكون المعلومات المجمعة لا تعكس الواقع المعيش.

لهذا من الأفضل أداء هذا العمل(في بعض الحالات) من دون علم الموظف القائم بالعمل وهذا لا يعني أنه في حالات معينة يمكن أن يتفاعل محلل النظم مع الشخص المعني وأن يوجه الأسئلة أو يشاهد ويجمع بعض النماذج المستخدمة.

3.4.4 جمع وتحليل العينات والوثائق Document Gathering & Sampling

تعتبر طريقة جمع وتحليل النماذج Forms والوثائق Documents هامة أيضا وجدية، لأننا قد نجد طرقا أخرى تساعدنا على معرفة طريقة عمل النظام كما هي على أرض الواقع أو كما تم شرحها أثناء المقابلة والملاحظة.

لكن السؤال هو هل يعمل المستخدم بالطريقة المثالية والصحيحة المدونة بالقوانين واللوائح في شكل وثائق أو لا؟

وللإجابة عن السؤال أقول: عند جمع عينات من الوثائق الخاصة باللوائح والملفات والنماذج والتقارير والمخططات التنظيمية للمؤسسة وتحليلها يمكن معرفة ومقارنة طريقة عمل النظام بالطريقة المثالية وطريقة عمل النظام بالطريقة الفعلية.

وبالطبع سيستفيد محلل النظم من هذا عند تحديد المتطلبات للمنظومة الجديدة.

3.4.5 تصميم التطبيق المشترك Joint Application Design (JAD)

بدأت هذه الطريقة في أواخر السبعينيات من قبل شركة IBM لغرض جمع كل من الزبون

(المدير ورؤساء الأقسام مثلا) والمستخدمين ومحللي النظم والمبرمجين ومقرر الجلسات في عملية تحليل النظام.

وانتشرت هذه الطريقة JAD بعد ذلك كأداة مهمة تستعمل في إعداد المنظومة وذلك بإشراك الزبون في معظم المراحل.

وفي مرحلة التحليل يتم عمل ورش عمل(جلسات) لجمع المتطلبات من قبل الذين لهم علاقة بالنظام (Stakeholders) في مكان واحد وفي وقت واحد ولمدة قد تستغرق أسبوعا.

وهي جلسات مكثفة لحل المشاكل، أو على الأقل معرفة السبب في صعوبة إيجاد الحلول.

إن مشاركة المستخدم في أغلب مراحل المشروع أصبح من المفاهيم الحديثة والمرغوبة. وقد تمّ إيجاد علاقة بين مشاركة المستخدم ورضاه عن المنظومة حين تسلمها، لأن أفضل الأفكار الإبداعية للمنتجات الجديدة وتحسينها تأتي عادة من الزبون وليس من معد المنظومة.

ويجب أخذ الملاحظة بأن مكان الاجتماع يجب أن يكون بعيدا من مكان عمل الزبون ويجب إعداد الحجرة التي ستعقد فيها الجلسات جيدا، بمعنى وجود السبورة البيضاء وجهاز حاسوب للمقرر وطابعة لطباعة الوثائق التي قد توزع على الحاضرين وجهاز عرض الشرائح باستخدام الحاسب.

أمّا بعد المكان فمردّه إلى الابتعاد عن الإزعاج والإرباك وزحمة العمل، ليتسنى التركيز على العمل الذي كلفوا به.

وتتلخص النقاط في هذه الجلسات في ما يلي:

1. لمحة على النظام الحالي والمشاكل التي تعترضه.
2. مناقشة لتصميم النماذج والتقارير للمنظومة المقترحة (واجهة المستخدم).
3. مناقشة حول وظائف المنظومة المقترحة وخصائص الجودة المطلوبة في هذه المنظومة.

3.4.6 تخطيط المتطلبات المشترك Joint Requirement Planning (JRP)

هذه الطريقة JRP لجمع المتطلبات تعتبر حالة خاصة من طريقةJAD وهي مصطلح خاص بالتحليل والمتطلبات، بينما JAD لجميع المراحل.

وهي جلسة عمل جماعية بعكس الطريقة التقليدية المتمثلة في: المقابلة التي يتم فيها اجتماع منفرد مع الزبون أو المستخدم.

فهذا الاجتماع الجماعي للأشخاص الذين لهم علاقة بالمنظومة الجديدة له هدف واحد يتمثل في التعرف على المشاكل التي تعترض النظام الحالي وتحليلها ومحاولة تحديد المتطلبات للمنظومة الجديدة.

وتتركز فكرة هذه الطريقة على مبدأ الإجماع والشورى في تحديد المشاكل والمتطلبات والأهداف، ويتم الاجتماع عادة في أماكن مجهزة لهذا الغرض.

وفي هذه الجلسات يمكن إثارة وتوليد أفكار لحل المشاكل التي تعترض النظام الحالي باستخدام طريقة إثارة الأفكار Brainstorming

3.4.7 العرض التجريبي Prototyping:

يعتبر العرض التجريبي Prototyping من أهم الطرق الفعالة والناجحة لجمع المتطلبات الوظيفيه فهي طريقة لعرض منظومة تجريبية (نسخة مبدئية من المنظومة المعلوماتية) ذات الوظائف المحدودة تبين قدرات المنظومة على أداء وظائفها.

ويتم تجهيز هذه المنظومة التجريبية في مدة وجيزة، وباستخدام لغات برمجة سهلة وسريعة التنفيذ، مثل بيسك المرئي أو برنامج أكسس لقواعد البيانات.

وباستخدام هذه الطريقة ظهر تحسن واضح في حل مشكلة المتطلبات مقارنة بالطرق الأخرى، لأن هذه الطريقة تمكّن من حصول بيئة تفاعلية بين معد المنظومة والمستخدم وجهاز الحاسوب ينتج من خلالها استنباط المتطلبات.

وهي أداة تواصل جيدة بين جميع الأطراف ذات العلاقة بالمنظومة، وتبين بشكل مبدئي شكل المنظومة النهائي، حيث نرى أنّ المتطلبات التي قد تمّ جمعها في المقابلة الشخصية قد تجسدت واقعيا في منظومة.

لهذا يتم إصدار عدة نسخ تباعا (عدة تصميات) نتيجة عرض المنظومة على المستخدم

وأخذ الملاحظات والاقتراحات منه لتنفيذها مرة أخرى وهكذا يتم تنقيح المتطلبات بهذه النسخ المتتالية.

هناك ملاحظة أخيرة بخصوص العرض التجريبي وهي أن استخدامه يتم عادة عندما لاتكون المتطلبات واضحة ومفهومة وعندما توجد صعوبة بين المحلل والمستخدم في تناول المتطلبات باستخدام الطرق الأخرى.

3.4.8 حالات الاستخدام والسيناريو Use Cases and Scenarios

تستخدم طريقة حالات الاستخدام Use cases للتحديد والتعرف على المتطلبات الوظيفة للمنظومة المزمع تنفيذها. وتسمى هذه الوظائف (المعاملات) بحالات الاستخدام.

وفي هذه الطريقة يقوم محلل النظم بالاستفادة من المقابلات الشخصية والملاحظة بالتعرف على الذين يقومون بأداء الوظائف ومن ثمّ رسم مخطط حالة الاستخدام UCD بمساعدة المستخدم.

ثم بعد ذلك يقوم المستخدمون بشرح كيفية أداء كل معاملة أو وظيفة خاصة بالمنظومة على شكل وصف نصي لكل حالة استخدام على حده ويسمى هذا النص بالسيناريو Scenarios ونلحق بهذا المخطط، وبالتالي الحصول على المتطلبات الوظيفية Functional requirements .

ويفضل استخدام ورش العمل Workshops في شكل جلسات يحضرها المستخدم والزبون ومحلل النظم وفريق العمل الآخر.

وهذه الطريقة تشجع عملية الإجماع حول المتطلبات ومن ميزاتها رخص تكاليف استعمالها.

وهي طريقة تفاعلية بين كل المستفيدين والمتأثرين بالمنظومة وبها يتم إدماج المستخدم في المنظومة منذ البداية.

3.4.9 جلسة توليد الأفكار Brainstorming

تعتبر طريقة جلسة توليد وإثارة أفكار Brainstorming ذات أهمية حيث يتم تكوين وجمع واقتباس المتطلبات من مجموعة من الأشخاص ذوي العلاقة بالمنظومة (Stakeholders) في مدة قصيرة وتوليد عدة أفكار بخصوص المزايا المرتقبة من المنظومة.

ثمّ بعد ذلك يتم تصنيفها حسب الأهمية، ويتم أيضا اكتشاف متطلبات مخفية ولم يتم التعرف عليها بالطرق الأخرى. وهذه الطريقة تشجع على التفكير المنظم وإعطاء فرصة للإبداع.

3.4.10 البحث والتطبيقات المشابهة Research & Similar Applications

تعتبر طريقة البحث Research من الطرق المهمة لتجهيز محلل النظم للعمل في المنظومات الجديدة التي ليس له سابق خبرة بها ولا بمصطلحاتها ومفرداتها وقد سبق للمؤلف أن عاصر إعداد منظومة كان البحث أحد الأسباب في نجاحها.

والسبب أن المقابلة الشخصية للزبون هي الانطباع الأول الذي يأخذه الزبون على إمكانيات محلل النظم وكلما كان محلل النظم فاهما لطريقة العمل وواعيا للمصطلحات المستخدمة (مثلا المصطلحات المصرفية) زادت ثقة الزبون في معد المنظومة، ومحلل النظم هو الواجهة الأمامية لذلك.

يتضمن البحث عادة المكتبة والانترنت ودراسة السوق ومراجعة تطبيقات مشابهة Similar Applications في الشركات وأقسام الحاسوب بالجامعات والمعاهد التقنية .

في آخر هذا البند أود أن أركّز على نقطة أساسية وجوهرية وهي أن إلحاق المستخدم User بفريق إعداد المنظومة من بداية مراحل إعداد المنظومة يعتبر من الممارسات الإيجابية والممتازة للحصول على متطلبات كاملة وصحيحة والتي بدورها ستساعد على نجاح المنظومة.

والعكس صحيح: أي أن عدم إلحاقه سيكون أحد العوامل لانتكاسة المنظومة وبالتالي عدم ضمان نجاحها.

3.5 تحليل المتطلبات Requirement Analysis

إن فكرة التحليل أساسا هي تقييم احتياجات المستخدم للوصول إلى تعريف محدد للمتطلبات البرمجية المراد تجهيزها.

ونعني بتحليل المتطلبات: عملية تفكيك وتجزئة المتطلبات العامة (العالية المستوى) وتحويلها إلى متطلبات وظيفية تفصيلية (متدنية المستوى) حيث يتم إستخدام أدوات مناسبة لتمثيلها ونمذجتها Modeling.

وكملاحظة في هذا السياق يجب أن أقول أنه يجب تصنيف وترتيب هذه المتطلبات ليتم تنفيذها حسب الأهمية.

وهذا التصنيف للمتطلبات يمكن أن يأخذ شكل:

- متطلبات ضرورية.
- متطلبات مشروطة.
- متطلبات اختيارية.

وعند تحليل المتطلبات أي نمذجتها نستخدم الأدوات بناء على المنهجية التي يتم اختيارها لعملية التحليل للمنظومة: وتوجد منهجيتان مشهورتان على نطاق واسع وهما:

3.5.1 المنهجية الهيكلية Structured Methodology

هذه المنهجية تركز أكثر على وظائف المنظومة (المعالجة)، حيث يتم استخدام الأدوات المستخدمة في التحليل الهيكلي Structured Analysis لتمثيل ووصف نمذجة هذه الوظائف.

ومن هذه الأدوات :

- مخطط انسياب البيانات Data Flow Diagram (DFD)

وهو مخطط هيكلي رسومي يبين صورة لحركة انسياب البيانات داخل النظام بين مخازن البيانات والمعالجة والكيانات الخارجية.

وهناك نوعان من مخطط DFD:

الأول: يستخدم لتمثيل النظام كما هو على أرض الواقع مشتملا على المعلومات والمواد ويسمى المخطط الانسيابي المادي Physical DFD وهو وسيلة تفاهم بين المستخدم ومحلل النظم حتى يفهم محلل النظم الدورة المستندية وتحرك المواد عبر النظام ليتسنى له فهم عمل النظام وبالتالي تزال عملية الغموض لديه.

الثاني: يستخدم ليكون أساسا لمرحلة التصميم ويسمى المخطط الانسيابي المعنوي Logical DFD ويشتمل على انسياب البيانات فقط محذوفا منه انتقال المواد.

وعملية رسم مخطط انسياب البيانات تبدأ برسم مخطط عام وعالي المستوى (وبدون تفاصيل) يسمى المخطط البيئيContext Diagram يبين تفاعل المنظومة مع الكينونات الخارجية (البيئة الخارجية للمنظومة).

ثم يقوم محلل النظم بتفصيل المخطط البيئي ليشمل على مخططات تفصيلية تسمى DFD level 1 وDFD level 2 وهكذا تتم التجزئة تباعا.

ومن مزايا هذه التجزئة الهيكلية ازدياد سهولة فهم النظام أو المنظومة من خلال المناقشة حولها بين جميع الأطراف ذات العلاقة.

ويعتبر مخطط انسياب البيانات بسيط التكوين الاّ أنه أداة قوية لتمثيل(نمذجة) وظائف النظام(أو المنظومة).

ويعتقد العديد من محللي النظم أن هذا المخطط هو كل ما يحتاجونه لمعرفة التحليل الهيكلي.

والحق أنه بدون استخدام أدوات أخرى مساعدة لعملية التحليل تصبح هذه الأداة غير ذات جدوى في حد ذاتها.

لهذا يجب جمع هذه الأداة مع أدوات أخرى مثل قاموس البيانات Data Dictionary والإنجليزية (العربية) الهيكلية Structured English ومخطط الكائنات العلائقية Entity Relationship Diagram وشجرة القرار Decision Tree وغيرها.

■ قاموس البيانات Data Dictionary

يمكن تعريف أداة قاموس البيانات Data Dictionary بأنه قائمة List لكل عناصر البيانات Data Elements الخاصة بالمنظومة، أو وصف لمخازن البيانات وانسيابها والموجودة في مخطط DFD.

لهذا يجب تجهيزها بعد إعداد هذا المخطط السالف الذكر.

ونظرا لأهمية هذه الأداة لتحديد متطلبات البيانات Data Requirements الخاصة بالمستخدم، فإنها يجب أن تكون دقيقة وواضحة لكي تساعد على التفاهم المشترك بين كل من المستخدم ومحلل النظم والمصمم والمبرمج وإبعاد أي لبس حول مدخلات ومخرجات المنظومة.

وتعتبر هذه الأداة من الأساسات التي سيستفاد منها في تصميم قاعدة البيانات في مرحلة التصميم.

■ الإنجليزية الهيكلية Structure English

تعتبر الإنجليزية الهيكلية Structure English أداة تحليل نصية تستخدم جزء محدود من اللغة الإنجليزية لتوضيح الخطوات المراد أداؤها لوظائف (عمليات) منظومة معلومات.

وتستخدم أفعال مثل :read، write، print، sort، add، subtract.....etc وعناصر بيانات مثل Customer-no، item-no، price (مبينة في قاموس البيانات) وتركيبات البرمجة الهيكلية مثل If-then-else أو While-Do وعادة ما تستخدم الإنجليزية الهيكلية بعد رسم مخطط انسياب البيانات لتوضيح كل معالجة Processes في DFD.

وتعتبر الإنجليزية الهيكلية أداة تواصل مع المستخدم لإزالة الغموض حول كل عملية.

وهي أساس كتابة شبه الشفرة Pseudocode في مرحلة التصميم.

■ جدول القرارDecision Table

يعتبر جدول القرارDecision Table أداة تحليل في شكل جدول(مصفوفة) يبين الأفعال Actions المحتملة بناء على شروط Conditions معينة.

وتساعد هذه الأداة في توضيح القرارات التي تستخدم في الحالات المعقدة.

ويستخدم محلل النظم هذه الأداة لتبيين سياسة عمل النظام مثل:

- سياسة التخفيضات في نظم المبيعات
- سياسة تنسيب طلبة الثانوية للجامعات والمعاهد.
- سياسة تقييم المستوى الأكاديمي للطالب بناء على أدائه في الامتحانات والواجبات المدرسية.

ولتوضيح فكرة عمل هذه الأداة نورد المثال التالي:

شركة الأمل لتجارة الاثاث تمنح تخفيضا (خصما) على قيمة المشتريات على منتجاتها للزبائن حسب سياسة الشركة كالآتي :

1. إذا كان الزبون يدفع بالحاضر ومجمل مشترياته تفوق 3000 دينار فتكون نسبة التخفيض 10 ٪.
2. إذا كان الزبون يدفع بالحاضر ومجمل مشترياته أقل أو يساوي 3000 دينار فتكون نسبة التخفيض 6 ٪.

فمن هذه الحالات يمكن رسم جدول القرار. (انظر مثلًا كتاب تحليل النظم للمؤلف)

■ شجرة القرار Decision Tree

شجرة القرار Decision Tree أداة تحليل على شكل شجرة تبين الحالات (الشروط) Conditions والأفعال Actions ذات العلاقة بهذه الشروط وهي تبين سياسة عمل النظام. وهي تشبه إلى حد بعيد عمل جدول القرار الآ أن تفرعاتها يجب أن تكون محدودة وهي بديل لجدول القرار في النظم الغير معقدة.

وفي الأداة السابقة (جدول القرار) عرضنا مثال شركة الأثاث وسياستها في التخفيض. ونظرا لبساطة هذا النظام فمن الأفضل توضيح هذه السياسة باستخدام شجرة القرار.

■ مخطط الكائنات العلائقية Entity Relationship Diagram (ERD)

يعتبر مخطط الكائنات العلائقية ERD (Entity Relationship Diagram) بداية صحيحة من قبل محلل النظم لفهم متطلبات البيانات Data Requirements الخاصة بالمنظومة تحت الإعداد.

ويتكون مخطط الكائنات العلائقية من أشكال هندسية تشبه المخطط الانسيابي تبين الكائنات Entities والعلاقة Relationships بين هذه الكائنات وأيضا الخصائص Attributes (عناصر البيانات) لكل كائن أو علاقة.

ويعتبر هذا المخطط أيضا أساسا لتصميم قاعدة البيانات.

وهو أداة تواصل بين محلل النظم والمستخدم لفهم وتدوين عناصر البيانات المستخدمة وانتمائها للكيانات.

وفي الوقت الحاضر وباستخدام العرض التجريبي Prototyping أصبح الحصول على عناصر البيانات عملا ميسرا.

3.5.2 المنهجية الشيئية Object-Oriented Methodology

يمثل التحليل الشيئي Object-Oriented Analysis تغيرا دراميا مقارنة بالتحليل الهيكلي حيث يتم التعامل مع النظام على أساس أنه مجموعة من الكائنات (المادية والمعنوية).

أمّا التحليل الهيكلي فيعتبر البيانات منفصلة عن العمليات التي تحصل على هذه البيانات.

بمعنى أن البيانات ليس لها أهمية بالغة في التحليل الهيكلي Structured Analysis.

حيث يتم تقسيم المنظومة إلى وظائف رئيسية وتجزأ هذه الوظائف إلى وظائف فرعية وهكذا.

أمّا غرض التحليل الشيئي OOA فهو ربط البيانات والعمليات في مكان واحد وهو الكائن Object أو الفصيلة Class علما بأن الكائن حالة خاصة من الفصيلة.

وفي هذا النوع من التحليل يمكن تعريف الفصائل والعلاقات بين هذه الفصائل والنظام.

ويجب اتباع الخطوات التالية للحصول على هذا التحليل :

1. يبدأ محلل النظم في الحصول على بعض المتطلبات من الزبون والتي من أهمها المتطلبات الوظيفية، ويستخدم مخطط استخدام الحالة Use Case Diagram (UCD) كأداة هامة لتحديد هذه المتطلبات. ويستخدم السيناريو Scenario النصي لوصف كل حالة استخدام.
2. التعرف على الفصائل Classes الخاصة بالنظام (أيضا الخصائص Attributes والطرق Methods لكل فصيلة).
3. رسم مخطط الفصائل Class Diagram ويتم ذلك إمّا يدويا أو بإستخدام أحدى أدوات Case.
4. رسم مخطط السلسلة Sequence Diagram ليعبر عن وصف تفصيلي لكل حالة استخدام.

في هذا البند نقوم بشرح ثلاث مخططات شيئية تستخدم في مرحلة التحليل:

■ مخطط حالة الاستخدام Use Case Diagram UCD

يعتبر مخطط حالة الاستخدام UCD أداة تحليل شيئية مهمة لتوضيح المتطلبات الوظيفية للنظام. ويتكون من أشكال هندسية تعبر عن حالة الاستخدام Use case وهي المعاملة أو الوظيفة التي يؤديها النظام، والممثل أو الفاعل Actor وهو الذي يقوم بأداء هذه المعاملة (حالة الاستخدام).

■ مخطط الفصيلة Class Diagram

مخطط الفصيلة Class Diagram هو أداة تحليل شيئي رسومي يبين هيكلية الكائنات الساكنة للنظام. وهذه الهيكلية تبين فصائل الكائنات والعلاقة بين هذه الكائنات. ويبين مخطط الفصيلة الحركة الساكنة للمنظومة الشيئية.

■ مخطط السلسلة Sequence Diagram

يعتبر مخطط السلسلة Sequence Diagram أداة تحليل شيئية تبين الكائنات والتواصل بين هذه الكائنات باستخدام الرسائل المتبادلة بينهم عند تنفيذ حالة الاستخدام.

ويبين مخطط السلسلة الحركة المتحركة (الديناميكية) للمنظومة الشيئية.

3.5.5 منهجية أجل Agile Methodology

تعتمد منهجية Agile على اختيار أفضل الأدوات والطرق المناسبة لأداء المهام الخاصة بالمنظومة.

وتعتمد على استعمال عدة منهجيات في المشروع الواحد مثل استخدام الأداة DFD مع الأداة UCD في نفس المشروع أي المنهجية الهيكلية والمنهجية الشيئية.

وبعبارة أخرى يمكن استخدام منهجيات وطرق مختلفة والهدف هو تسريع عملية إعداد المنظومة. وهذه المنهجية تفاضل بين الإنتاجية والجودة.

3.5.4 منهجية إطار عمل الحلول Ms-Solution Framework Methodology (MSF)

في هذه المنهجية MSF التي تستخدمها شركة Microsoft في إعداد منظوماتها يستطيع محلل النظم أن يصمم عدة نماذج Models منها على سبيل المثال لا الحصر:

■ نموذج المعالجة Process Model

■ نموذج البيانات Data Model

■ نموذج إدارة المخاطرة Risk Management Model

وكل نموذج يساهم في تحليل وتصميم المنظومة تحت الإعداد.

وتستخدم منهجية MSF أدوات OOA مثل UCD وأدوات CASE.

ونعيد هنا القول أنه: للحصول على منظومة ذات جودة، يجب إدماج وإلحاق المستخدم جنبا إلى جنب مع معد المنظومة في هذه المرحلة المهمة ألا وهي التحليل.

3.6 المواصفات Specifications

في هذا النشاط يتم كتابة وتجهيز وثيقة هامة من وثائق المنظومة في نهاية مرحلة التحليل وتسمى وثيقة مواصفات المتطلبات Requirement Specification Document والتي تشتمل على كل متطلبات المنظومة المقترحة.

وتلعب وثيقة المتطلبات دورا مهما في دورة حياة المنظومة، لأنها تقودنا إلى مراحل التصميم والتنفيذ، وتعتبر أساسا للتعاقد بين الزبون ومعد المنظومة.

ولقد ثبت أن حوالي 85 ٪ من أخطاء البرمجيات كان مردّه إلى المتطلبات ومشاكلها والتي هي:

■ 49 ٪ افتراضات تتعلق بمتطلبات غير صحيحة.

■ 29 ٪ متطلبات محذوفة (غير معلنة).

■ 13 ٪ متطلبات متضاربة.

■ 5 ٪ متطلبات غامضة وغير واضحة.

وقد ثبت من الإحصائيات بسبب مشكلة تحديد المتطلبات أيضا أن حوالي 30 ٪ من المشاريع يتم إلغاؤها قبل أن تنتهي، وأن حوالي 50 ٪ من المشاريع تكلف الضعف من التقديرات الأولية.

ولكتابة وثيقة مواصفات المتطلبات يمكن استخدام معايير مؤسسات ذات علاقة مثل:

1993-ACM,IEEE، ANCI 830

3.6.1 خصائص مواصفات المتطلبات البرمجية:

تناول العالم Boehm (1984) خصائص مواصفات المتطلبات البرمجية الجيدة فيما يلي:

- كاملة Complete
- دقيقة وقابلة للقياس Measurable
- صحيحة Correct
- واضحة Unambiguous
- قابلة للاختبار Testable
- متناغمة (غير متضاربة) Consistent
- موجزة ومحددة Concise
- قابلة للتحقق Verifiable
- سهولة التعديل Changeable
- بدون علاقة بالتصميم Design-free

وقبل أن نشرع في هذه الوثيقة، نشرح معنى وأهداف التوثيق :

3.6.2 التوثيق Documentation

يعتبر التوثيق عنصرا مهما في إعداد البرمجيات واستمرار عملها بعد الإعداد.

ويمكن تعريف التوثيق بأنه مجموعة أوصاف نصية ورسومية وشروح للمنتوج البرمجي (المنظومة البرمجية).

وقد يشمل التوثيق ما يلي:

- سرد أو نص Narratives
- مخططات Charts
- جداول Tables
- الصوت Voice
- قصاصات فيديو Video clips
- صور متحركة Animations
- تعليقات في البرنامج Comments in program

ويمكن أن تكون الوثيقة على شكل ورقة أو تكون مخزنة في الحاسوب.

3.6.3 أهداف ووظائف التوثيق :

يؤدي التوثيق الوظائف التالية:

1. مرجع تاريخي.
2. مرجع إرشادي وتوضيحي.
3. متابعة جودة المنتوج البرمجي.
4. التواصل بين مراحل إعداد المنظومة.
5. التواصل بين المهام داخل المرحلة الواحدة.
6. اتفاق بين المستخدم أو الزبون ومعد المنظومة.

3.6.4 استخدام التوثيق:

يستخدم توثيق المعد بصورة عامة من قبل:

- الإدارة لغرض المراجعة.

- ■ القائمين على صيانة البرنامج.
- ■ فريق التفتيش.
- ■ فريق المراجعة غير الرسمية من قبل زملاء العمل.
- ■ موظفي التحقق والمصادقة.

نستعرض الآن محتويات وثيقة مواصفات المتطلبات التي يجب أن يعدها محلل النظم في نهاية مرحلة التحليل ولتتم مراجعتها من قبل الإدارة لاتخاذ أحد القرارات الآتية:

- ■ الاستمرار في المشروع وتنفيذ المرحلة التالية وهي التصميم.
- ■ وقف استمرار المشروع.
- ■ إجراء بعض التعديلات ثم الاستمرار في المشروع.

3.6.5 وثيقة تحديد المتطلبات Requirement Specification Document،

تعريف وثيقة تحديد المتطلبات:

هي وثيقة يتم إعدادها في نهاية مرحلة التحليل تتضمن وظائف المنظومة المراد تنفيذها وخصائص الجودة المتعلقة بها. وهذه الوثيقة يجب أن تكون صحيحة ودقيقة وكاملة ومتناسقة وقابلة للقياس والاختبار.

يبين الشكل (3.6.1) البنود التي تشملها عادة هذه الوثيقة.

---

a) Introduction مقدمة

1. **Overall description وصف عام:**

- ■ problem definition تعريف المسألة
- ■ objectives of the system الأهداف
- ■ interfaces of the system البينيات

- ■ scope of the system حدود المنظومة
- ■ constraints of the system القيود

2. **الوصف الوظيفي Functional description:**

- ■ list of system functions قائمة الوظائف
- ■ Narrative for each function وصف كل وظيفة

b) Data/ Information description وصف البيانات:

- ■ Entity relationship diagram مخطط (ERD)
- ■ Data dictionary قاموس البيانات

c) Process/ logic description وصف المعالجة والمنطق:

- ■ DFD (Data Flow Diagram) مخطط انسياب البيانات
- ■ UCD (Use Case Diagram) مخطط استخدام الحالة
- ■ Structured English الإنجليزية الهيكلية
- ■ Decision tree شجرة القرار
- ■ Decision table جدول القرار

d) Performance Requirements متطلبات الأداء:

- ■ Response Time زمن الاستجابة
- ■ Memory الذاكرة

e) Validation / Acceptance Criteria معيار التحقق والقبول:

- ■ Types of test أنواع الاختبارات
- ■ Quality attributes required خصائص الجودة المطلوبة

■ Deliverables — البنود القابلة للتسليم

f) Solution Strategy إستراتيجية الحل:

■ on-line/off-line — على الخط أم خارج الخط

■ Graphic / text — رسومات أم نص

■ database / files — قاعدة بيانات أم ملفات

الشكل (3.6.1) بنود وثيقة مواصفات المتطلبات

ملاحظات عن وثيقة مواصفات المتطلبات:

1. تصف مشاكل وليس حلولا.
2. هي منتوج وليس عملية معالجة.
3. وثيقة بين الزبون والمحلل وتستخدم فيما بعد في التصميم.
4. تقوم بتحويل الاحتياجات إلى متطلبات.
5. يجب مراجعتها من قبل المستخدم ومعد المنظومة.
6. تبين ما هو المتوقع من المنظومة وليس كيفية العمل.

3.7 المصادقة Validation

يعتبر هذا النشاط مهما للغاية يهدف في النهاية إلى التأكد Confirmation من أن مواصفات المتطلبات التي تمّ تجهيزها في البند السابق تتوافق مع المعايير Standards في كتابة وثيقة المتطلبات وجاهزة لأن تكون أساسا لعملية التصميم في المرحلة اللاحقة لمرحلة التحليل.

ويستخدم في هذا التحقق والاختبار عدة طرق للفحص والمراجعة والتأكد والتي منها:

■ الفحص Inspection Formal من قبل متمرسين في الكشف عن الأخطاء لهم سابق خبرة في أخطاء سابقة لمنظومات مشابهة وقديمة.

■ المراجعة السريعة Walkthrough عن طريق زميل في فريق إعداد المنظومة.

■ التحقق Verification من قبل جهة مستقلة مثل مكتب استشاري أو محلل متمرس من أن أهداف واحتياجات الزبون قد تمّت ترجمتها في شكل متطلبات.

■ المراجعة النهائية Review في نهاية مرحلة التحليل بحضور الزبون ورئيس وأعضاء فريق المنظومة لاتخاذ القرار النهائي بخصوص وثيقة المتطلبات.

ونلاحظ هنا أننا نقوم باختبار المتطلبات لإيجاد الأخطاء ولكن ليس على جهاز الحاسوب بل على الوثائق علما بأن التعرف على الأخطاء في بداية مراحل المشروع يقلل التكلفة.

3.8 إدارة المتطلبات Requirement Management

يرتكز هذا النشاط على أحد المخاطر أو المشاكل التي قد تعترض المشروع أثناء تنفيذ جميع مراحله ألا وهو التغير في المتطلبات في حياة المشروع وقبل البدء في تشغيل المنظومة على ارض الواقع.

وهذه المشاكل تسبب الإرباك وأيضا المتاعب لفريق إعداد المنظومة.

وللمساعدة في التسهيل والتعامل مع مشاكل التغير في المتطلبات، تمّ استخدام هذا النشاط ليكمل النشاطات السالفة الذكر.

وهذا النشاط يسمى إدارة المتطلبات Requirement Management.

إن إدارة المتطلبات: هي دراسة واستخدام الإجراءات والسياسات والعمليات التي تحكم كيفية التعامل مع التغير في المتطلبات، وبمعنى أدق:

1. كيفية تقديم مستند طلب تغيير Change Request
2. كيفية تحليل هذا الطلب ومعرفة تأثيره على التكاليف والجدول الزمني وحدود المشروع
3. كيفية المصادقة والموافقة على إجراء التغيير أو رفضه.
4. كيفية تنفيذ التغيير بعد أخذ الموافقة عليه.

ويهتم هذا النشاط في هندسة المتطلبات أيضا بالتخطيط Planning والمتابعة Controlling لنشاطات جميع المتطلبات والتحليل والمواصفات والتحقق.

ومن المهام الإدارية الخاصة بإدارة المتطلبات Requirement Management مايلي:

- إدارة النسخ الخاصة بالمنظومة والتغييرManaging versions and change
- تخزين خصائص المتطلباتStoring requirement Attributes
- التواصل مع الذين لهم علاقة بالمنظومة Stakeholders

وتوجد برمجيات جاهزة لإدارة المتطلباتِ Automated Requirement Management من قبل شركات متخصصة . ومن أبرز هذه البرمجيات:

- Doors
- Requisite Pro
- RTM Workshop
- Caliber-RM

ونظرا لأهمية المتطلبات والتعامل معها، فقد أنشأت بعض الشركات إدارة تعهد إليها بمتابعة التغييرات التي تحدث في المتطلبات ومتابعة إصدار النسخ والإصدرات لهذه البرمجيات والتي تسمى إدارة مكوّنات البرمجيات.

ولدراسة هذا الموضوع بأكثر تفصيل ندرس في البنود التالية موضوعا ذا علاقة بالتغيير في المتطلبات وإصدار النسخ وهو إدارة مكوّنات البرمجيات.

3.9 إدارة مكوّنات البرمجيات Software Configuration Management (SCM)

توجد في شركات إعداد البرمجيات إدارة تعنى بالتعامل مع التغيير تسمى:

إدارة المكوّنات البرمجية Software Configuration Management:

وتتعامل هذه الإدارة بالخصوص مع مشكلة التغير في المتطلبات من قبل المستخدمين أو معدي البرامج.

وهذا يعنى أن مدير إدارة المكوّنات البرمجيةSoftware configuration manager مسؤول عن استلام المنظومة من قبل مدير المشروع ومتابعتها مع الزبون طول مدة تشغيلها ومسؤول أيضا عن تقديم الدعم الفني للمنظومة طول فترة استعمالها.

قارن بين مدير المشروع البرمجي ومدير المكونات البرمجية؟

وبعبارة أخرى فإن للمنظومة مسؤولين هما:

- مدير المشروع: وهو مسؤول عن إعداد المنظومة من بداية مرحلة التخطيط إلى انتهاء مرحلة الاختبار(اختبار القبول) وتسليمها للزبون.

ماهي المهام الاساسية لمدير المكونات البرمجية

- مدير المكوّنات البرمجية Software Configuration Manager وهو مسؤول عن مرحلة الصيانة وإعداد الإصدارات والنسخ للمنظومة. ويتولى المهام التالية :
  - يتابع الإصدارات البرمجية software versions.
  - يقرر متى يجب إجراء التغيير.
  - يتأكد من أن الإصدارات (النسخ) تصدر للزبائن المناسبين في الوقت المناسب.
  - يتم تدوين طلبات التغيير.

3.10 لماذا نحتاج إلى إدارة مكوّنات البرمجيات؟

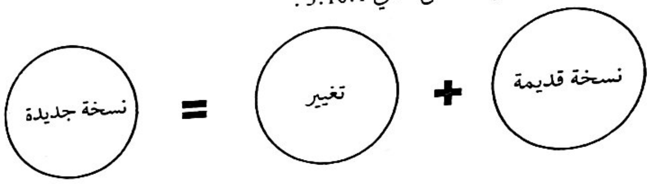
تعيش البرمجيات(المنظومات) وخاصة الكبيرة منها زمنا طويلا بعد استلامها.

لذلك يجب أن تكون قابلة للتغيير حتى تواكب كل ما هو جديد.

وفي عالم إعداد البرمجيات يتم عمل ما يسمى بالإصدارات Releases + Versions لنفس المنظومات

وكمثال على ذلك نجد أن برنامج رواتب مكتوب بلغة كوبول تحت نظام دوس في الثمانيات تم استبداله سنة 2000 بإصدار جديد باستعمال لغة دلفي تحت نظام ويندوز 2000.

فالإصدار الجديد عبارة عن الإصدار القديم مضافا إليه التعديلات(التغيرات) المستحدثة أو الجديدة كما هو مبين في الشكل التالي 3.10.1 :



الشكل (3.10.1) النسخ (أو الإصدار) والتغيير

> **ملاحظة: في حالة حدوث تغيير أثناء تشغيل المنظومة، يجب تجهيز إصدار جديد متضمنا التغيرات المناسبة، وهذا الإصدار يتم إعداده من قبل إدارة تعنى بمتابعة المنظومات بعد تسليمها للزبون.**

ماهي عناصر المكونات البرمجية

والأشياء المراد تغيرها تسمى عناصر المكوّنات البرمجية Configuration Management Items وتتضمن الآتي:

- برامج Programs
- بيانات Data
- وثائق Documents

3.11 نشاطات إدارة المكونات: ماهي أنواع نشاطات إدارة المكونات البرمجية

تقوم إدارة المكوّنات البرمجية بأداء المهام الآتية:

- تخطيط التغير.
- التحكم في التغير ومتابعته.
- إدارة الإصدارات والنسخ الجديدة.

وعند التخطيط يجب أن يكون لدينا قاعدة بيانات لتدوين معلومات المكوّنات البرمجية مثل:

- أسماء الزبائن الذين تحصلوا على إصدارات معينة من المنظومة.
- العتاد ونظام التشغيل المستخدم لإصدار معين.
- تاريخ وعدد الإصدارات لمنظومة معينة.
- عدد طلبات التغير التي تحت الإجراء لإصدار معين.

ويعتبر مصطلح Change (التغير) من المصطلحات الأساسية في إدارة مكوّنات البرمجيات (أو ما يسمى إدارة متابعة المنظومات) ويجب أن يمر هذا التغير عبر الخطوات التالية:
يمر التغير change بعدة خطوات ماهي

- يجب أن يتم تحديده Identified
- يجب أن يتم تحليله Analyzed
- يجب أن يتم تدوينه Recorded
- يجب أن يتم تنفيذه Implemented
- يجب أن تتم متابعته Controlled
- يجب إعداد تقرير حول التغير. Reported

عدد أنواع التغير

3.12 أنواع التغيير Types of changes

هناك عدة أنواع ذات علاقة بالتغير للمنظومة البرمجية والتي منها:

- التغير في المتطلبات.
- التغير في التقنية المستخدمة.
- تغيرات في بنية العمل.
- التغير بسبب الأخطاء.

ملاحظة هامة: لاحظ أن إدارة المكوّنات البرمجية SCM ذات علاقة وطيدة بمرحلة الصيانة أي بعد تسليم المنظومة إلى الزبون والبدء في تشغيلها بسبب وجود العديد من التغيرات التي تحدث أثناء دخولها للواقع والمحك العملي.

أمثلة على التغيير Examples of change

ومن الأمثلة العملية لبعض التغيرات التي قد تحدث مايلي:

- رغبة الزبون في تعديل المتطلبات الخاصة بالتقارير التى تعدّها المنظومة.
- رغبة معدي البرمجيات في تعديل الأساليب الفنية المستخدمة.
- التغيير من نظام تشغيل إلى آخر (مثلا من ويندوز إلى لينكس).

لهذا يجب أن تكون جميع المنتجات البرمجية: البرامج والبيانات والتوثيق تحت متابعة إدارة المكوّنات البرمجية.

وهذا يعني أن وثيقة توصيف المتطلبات، ووثيقة توصيف التصميم، وشفرة المصدر ..الخ يجب أن تكون في حوزة إدارة المكوّنات البرمجية SCM للتعامل معها بعد استلام الزبون للمنظومة والبدء في تشغيلها.

3.13 الخط الرئيسي Base-line

يستخدم هذا المصطلح كثيرا أثناء إعداد البرمجيات وهو ما يشبه وثيقة الخريطة المعمارية النهائية المعدة من قبل المهندس قبل الشروع في البناء، أي بعد إجراء العديد من التغيرات من قبل الزبون وقبل الاستقرار النهائي على الشكل الذي سيكون عليه المبنى.

ويسمى شبيه هذا المصطلح في علم الهندسة المعمارية والمدنية بالطبعة الزرقاء blue print أما في علم هندسة البرمجيات فيسمى الخط الرئيسي Base-line.

وكقاعدة عامة يعتبر أي جزء (منتوج برمجي) يتم إعداده ومراجعته (مثلا وثيقة مواصفات المتطلبات) في نهاية أي مرحلة من مراحل إعداد المنظومة خطا رئيسيا.

أي أن الخط الرئيسي هو وثيقة مواصفات نهاية مرحلة ما، ولايمكن إجراء أي تغيرات عليه إلا عن طريق قرار رسمي من الإدارة وتحت إجراءات محكمة.

أمثلة على الخط الرئيسي:

1. وثيقة مواصفات المتطلبات.
2. وثيقة مواصفات التصميم.
3. شفرة المصدر.
4. دليل المستخدم.

3.14 عملية التحكم في التغيير Change Control Process

يجب أن تتبع أي عملية تغيير الخطوات التالية:

1. تسليم طلب التغيير Change Request إلى مدير المكوّنات البرمجية.
2. تقوم إدارة المكوّنات البرمجية باتخاذ قرار بشأن قبول أو رفض طلب التغيير.
3. تنفيذ العناصر المطلوب تنفيذها والمدونة في طلب التغيير(في حالة قبول طلب التغيير).

كتابة التقارير Reporting

عندما نتحدث عن التغيير في المنظومة البرمجية يجب أن نتحدث عن إعداد التقارير التي تتعلق بهذا التغيير . بمعنى أننا يجب أن نعدّ تقارير تبين الوضع الحالي للمنظومة والوضع التي آلت إليه بعد إحداث التغيير بما يضمن جودة المنتوج البرمجي.

وتسمى هذه التقارير بتقارير الحالة status reports.

وتقارير الحالة عادة ما توفر معلومات إلى من يهمهم الأمر بالتغيير الذي حدث.

3.15 أسئلة للمراجعة

1. ما هي أنواع نشاطات إدارة المكونات البرمجية؟
2. قارن بين عمل مدير المشروع البرمجي ومدير المكونات البرمجية؟
3. عرف معنى مصطلح هندسة المتطلبات البرمجية؟
4. عرّف مصطلح المتطلبات ؟ وما هي أنواعه؟
5. ماهي الأهداف التي ترمي إليها هندسة المتطلبات البرمجية؟
6. ماهي الأنشطة الخمس الرئيسية لهندسة المتطلبات؟
7. أذكر أربع طرق لاستنباط المتطلبات مع تبرير أهمها ؟
8. ما هي الخطوات التي يجب اتباعها للتعامل مع التغييرات في البرمجيات؟
9. اذكر أربعة أنواع للتغيرات في البرمجيات.
10. اذكر بعض الأمثلة للتغييرات التي عادة ما تحدث بعد تسليم البرمجيات.
11. اكتب الخطوات اللازمة في عملية تغيير البرمجيات.
12. لماذا يجب إعداد تقارير عن حالة تقدم المشروع عند إحداث أي تغيير في المنظومة ؟
13. ما هي المهام الأساسية لمدير المكونات البرمجية؟
14. ماهي عناصر المكونات البرمجيةSoftwae Configuration Items؟
15. يمر التغيير Change بعدة خطوات. ماهي؟
16. عدّد أنواع التغيير؟
17. ما معنى مصطلح Baseline؟
18. متى تصبح الوثيقة خطا رئيسيا base line ؟ وضح بمثال.

الفصل الرابع

التصميم المعماري

Architectural Design



4.1 مقدمة

يعتبر التصميم Design من أهم مراحل إعداد المنظومة البرمجية وأدقّها.

فهو عملية ترجمة وتحويل متطلبات المستخدم التى تمّ الاتفاق عليها بين محلل النظم والمستخدم (والمدونة في وثيقة المتطلبات) إلى تمثيل للمنظومة المراد إعدادها أي الحل على ورق، وتشبه عملية التصميم طريقة إعداد الخريطة المعمارية التي يعدّها المهندس قبل عملية البناء.

والهدف الأساسي من التصميم هو إعداد منظومة خالية من الأخطاء يتم الاعتماد عليها لأداء الوظائف المطلوبة وتكون بالطبع سهلة الاستخدام والصيانة.

ونحن عند التصميم يجب أن نأخذ اعتبارات للعناصر المهمة في المنظومة وهي:

- البيانات Data
- المعالجة Process
- المستخدم User وهو العنصر المهم في نجاح أو فشل المنظومة.

وتنقسم مرحلة التصميم إلى النشاطات Activities الآتية :

1. التصميم المعماري Architectural design
2. تصميم واجهة المستخدم User Interface Design
3. تصميم البيانات Data design
4. تصميم الإجراءات (الاجزاء البرمجية) Procedural design (Module)

4.2 التصميم المعماري

ويقصد بالتصميم المعمارى تجزئة المنظومة المراد إعدادها إلى مكونات (أجزاء برمجية) في شكل هرمي، والغرض هو تبسيط المنظومة المعقدة لأجل سهولة تصميم خوارزمياتها وتسهيل عملية توزيعها على المبرجين.

في هذا الباب سوف نستعرض بعض الأدوات الشائعة والمستخدمة في التصميم المعماري Architectural Design والتي منها المخطط الهيكلي Structure chart ومخطط ورنير Warnier-Orr.

ثم نتطرق إلى بعض المفاهيم العامة في التصميم ومعايير التصميم الجيد.

4.3 المخطط الهيكلي: Structure Chart

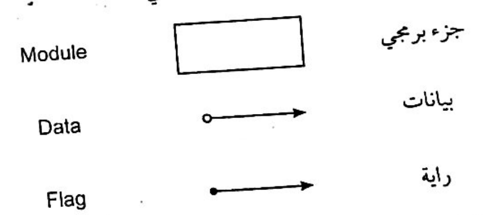
المخطط الهيكلي هو أداة تصميم رسومية توضح تركيبة (هيكلية) المنظومة في شكل هرمي.

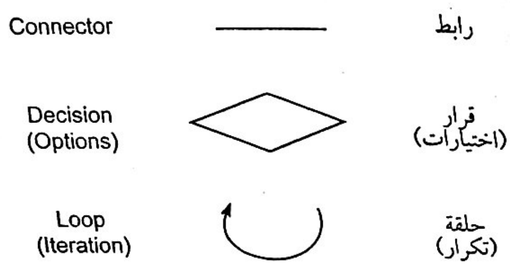
ويبين هذا المخطط العناصر الآتية :

- الهيكلية الشجرية أو الهرمية الكاملة للمنظومة كأجزاء برمجية(Modules).
- العلاقة بين هذه الأجزاء Relationship.
- البينيات (Interfaces) وهي البيانات الممررة بين هذه الأجزاء البرمجية.

مع ملاحظة أن هناك جزءا برمجيا للتحكم Control module سوّاق في أعلى مستوى وهو الذي يقود الأجزاء البرمجية الأخرى.

ويبين الشكل (4.3.1) رموز المخطط الهيكلي.





الشكل (4.3.1) رموز المخطط الهيكلي

ملاحظة: الرموز ذات العلاقة بتمرير البيانات ورايات التحكم تسمى عادة البينيات Interfaces بين الأجزاء البرمجية.

قواعد استعمال المخطط الهيكلي:

1. الأجزاء ذات المستوى المنخفض (في أسفل مستوى) تقوم بالمعالجة بينما الأجزاء ذات المستوى العالي تقوم بوظيفة التحكم.
2. استعمل (فعل أمر + مفعول به) مثل (احسب المرتب) لوصف العملية التي يقوم بها جزء برمجي معين .

و يمكن أن نستعمل الأفعال التالية لهذا الغرض:

- Get احصل على
- Read اقرأ
- Validate تحقق
- Compute احسب
- Check افحص
- Write اكتب

- Print اطبع
- Format شكّل
- Determine قرّر

يكون للأجزاء البرمجية ذات المستوى العالي أفعال عامة بلغة المستخدم تبين العملية ذات المستوى العالي مثل:

- Update حدّث
- Edit عدّل
- Process عالج
- Prepare جهز

ولكن الأجزاء ذات المستوى المنخفض يكون لها أفعال محددة للإشارة للمعالجة ذات المستوى المنخفض.

وهذه الأفعال تكون عادة مصطلحات حاسوب مثل:

- Format شكـّل
- Display اعرض
- Print اطبع
- Compute احسب
- Verify حقق
- Retrieve استرجع

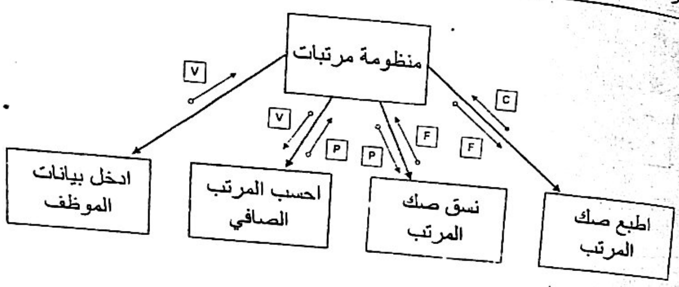
مثال:

دعنا نفحص المخطط الهيكلي التالي لمنظومة المرتبات كما مبين بالشكل (4.3.2)

نلاحظ في هذا الشكل أن منظومة المرتبات تتكون من الأجزاء البرمجية التالية:

- Input emp. Information ادخل معلومات الموظف
- Compute net pay احسب صافي المرتب
- Format pay check نسق (شكّل) صك المرتب
- Print pay check اطبع صك المرتب

ملاحظة: الأجزاء البرمجية يتم تنفيذها من اليسار إلى اليمين حسب الآتي:



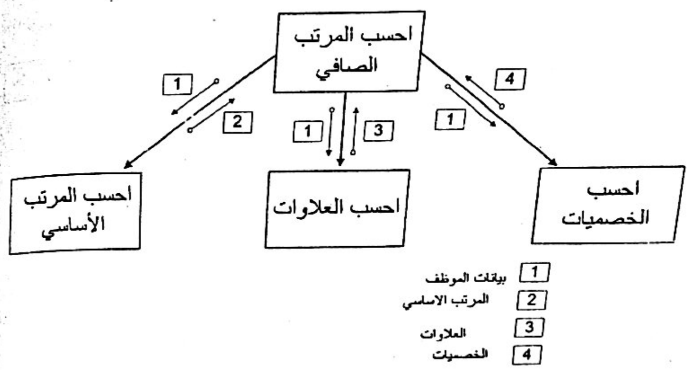
الشكل (4.3.2) مخطط هيكلي لمنظومة مرتبات

معاني الرموز في المخطط كالآتي:

- V = emp. Data معلومات الموظف
- P = Pay Check Information معلومات لصك المرتب
- F = Formatted Pay Check صك مرتب منسق
- C = Complete-task Flag راية انتهاء العمل

ويمكن تقسيم الجزء "احسب المرتب الصافي" إلى أجزاء فرعية أكثر تفصيلًا (كما مبين بالشكل 4.3.3) وهي :

- ■ Compute basic pay احسب المرتب الأساسي
- ■ Compute allowances احسب العلاوة
- ■ Compute deductions احسب الخصميات



الشكل (4.3.3) أجزاء فرعية للجزء «احسب المرتب الصافي»

Employee Data = 1	بيانات الموظف
basic pay = 2	المرتب الأساسي
Allowances = 3	العلاوات
Deductions = 4	الخصميات

في الشكل (4.3.3) تنفيذ مربعات من اليسار إلى اليمين تتابعيا .

ويوصف هذا النظام عادة بأنه نظام تحوّلي أو منظومة تحولية Transform Centered system.

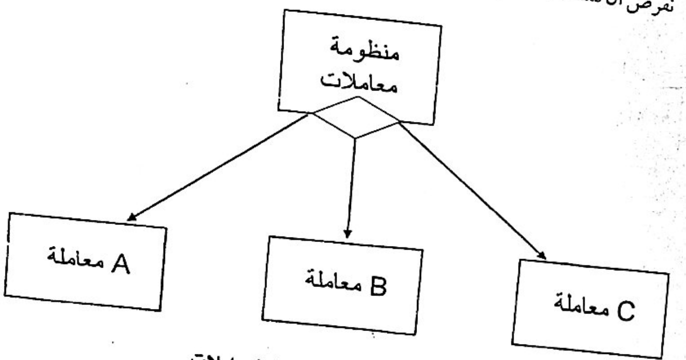
وفي التطبيقات التجارية عادة ما يستعمل نوع واحد فقط من المعاملات في الوقت الواحد.

وتوصف هذه المنظومات بأنها مركزة على المعاملة أو منظومة المعاملات Transaction-Centered System ومن أنواعها :

- ■ Inventory System منظومة المخزون
- ■ Banking System منظومة المصارف
- ■ Air-line Reservation System منظومة الحجز في الخطوط الجوية
- ■ Hotel system منظومة الفنادق
- ■ Sales system منظومة المبيعات

ولتوضيح هذا النوع (منظومة المعاملات) ندرس الشكل التالي:

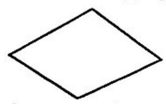
نفرض أن Trans اختصار لكلمة Transaction بمعنى معاملة (أي وظيفة تؤديها المنظومة).



الشكل (4.3.4) منظومة المعاملات

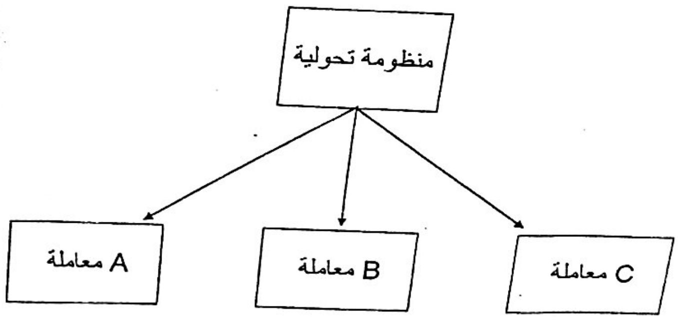
نلاحظ أن الجزء البرمجي المتحكم Driver أو السوّاق في الشكل (4.3.4) يقوم بتحويل التحكم لإحدى عمليات المعاملة بناء على المعلومات الواردة إلى هذا الجزء السواق.

ولاحظ أن المخطط المرتكز على المعاملة يستخدم دائما الرمز التالي :



هذا الرمز هو أداة ضبط التحكم حيث تتفرع منه عدة معاملات ولكن يتم تنفيذ واحدة فقط اعتمادا على نوع المعاملة المطلوبة.

أما المنظومة التحولية فهي كالمبينة في الشكل التالي حيث التنفيذ من اليسار الى اليمين تباعا:



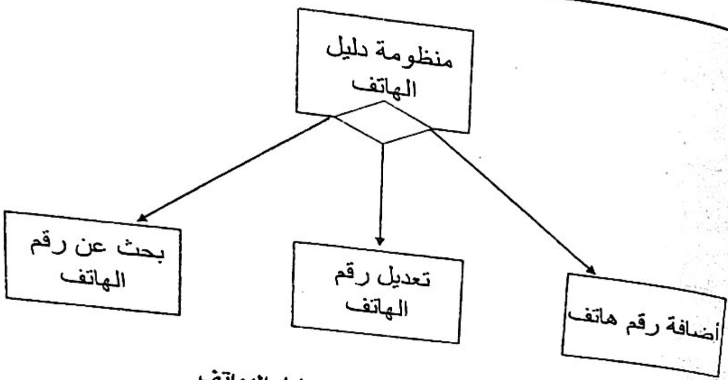
الشكل (4.3.5) منظومة تحولية

مثال:

ارسم المخطط الهيكلي لمنظومة لدليل الهواتف Telephone Directory بالحاسوب للوظائف التالية:

- Search for a phone number　البحث عن رقم هاتف
- Add new phone number　إضافة رقم هاتف جديد
- Change phone number　تغيير رقم هاتف

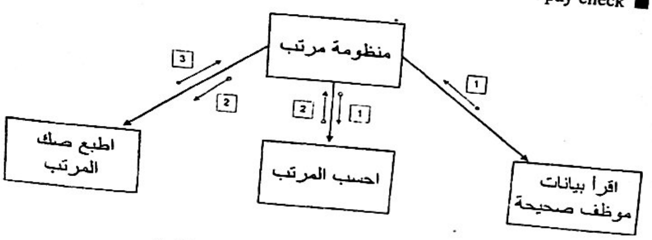
حيث يبين الشكل التالي المخطط المطلوب.



الشكل (4.3.6) منظومة دليل الهواتف

والآن دعنا نفحص المنظومة التحولية كما هو مبين بالشكل 4.3.6 (من اليمين الى اليسار) والذي يمثل نظام دفع الرواتب بالصك والمتكون من الأجزاء التالية :

- Get valid Emp data　ادخل بيانات موظف صحيحة
- Compute pay　احسب المرتب
- Print pay check　أطبع صك المرتب (من اليمين إلى اليسار)



الشكل (4.3.7) منظومة مرتبات تحولية

الرموز :

Valid Data :1 بيانات صحيحة

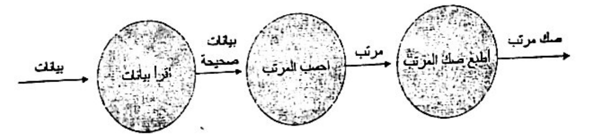
Pay :2 المرتب

End Flag :3 راية النهاية

والمنظومة في الشكل 4.2.7 يوضح مثالا لمنظومة تحولية حيث تتبع العمليات نفس المسار.

ويمكن توضيح ذلك أكثر بالشكل 4.2.8 وهو مخطط انسياب البيانات DFD أي Data Flow Diagram.

اقرأ بيانات صحيحة	Input valid data
احسب المرتب	Compute pay
اطبع صك المرتب	Print paycheck
المرتب	Pay
الصك	Paycheck



الشكل (4.3.8) لمنظومة تحولية DFD of Transform-Centered System

في هذا الشكل نرى أن البيانات تتحول من مرحلة إلى أخرى، حيث نبدأ بالحصول على بيانات صحيحة ثم نحسب قيمة المرتب، ثم أخيرا نطبع صك المرتب.

من دراسة الشكل DFD يستطيع المصمم أن يقرر التركيبة الفوقية المناسبة (التحويل أو المعاملة).

مثال:

حوّل منظومة الرواتب المبين بالشكل 4.3.8 من DFD إلى المخطط الهيكلي.

ملاحظة: عملية تحويل مخطط انسياب البيانات إلى المخطط الهيكلي تسمى بالتحليل التحويلي Transform Analysis.

الحل: لتحويل مخطط انسياب البيانات إلى مخطط هيكلي اتبع الخطوات التالية:

1. كوّن جزء تحكم (سواقة) تحت اسم معين ولتكن باسم (منظومة دفع الرواتب) في المخطط الهيكلي.
2. كل عملية Process في DFD يتم تغييرها إلى جزء برمجي (Module) في المخطط الهيكلي بنفس الاسم.
3. اربط هذه الأجزاء البرمجية بالسواقة مستخدما الوصلة.
4. انسيابات البيانات Data flows في مخطط DFD يجب أن تتحول إلى رموز بيانات أو رايات في المخطط الهيكلي.

ملاحظة: المخطط الهيكلي مبين في الشكل (4.3.7)

مثال:

افترض أننا نريد طباعة بيانات جميع الطلبة على الشاشة (منظومة الدراسة والامتحانات).

ارسم المخطط الهيكلي.

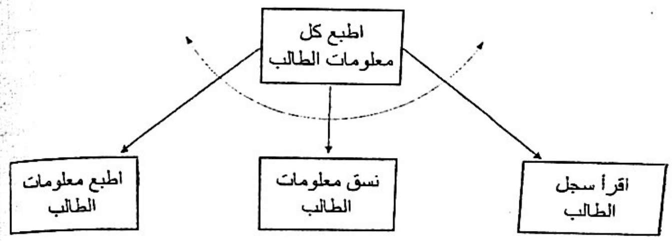
الحل:

يجب هنا استخدام رمز التكرار :



وذلك لغرض بيان أن العملية يجب أن تتكرر عددا من المرات كما يلزم.

المخطط المطلوب مبين بالشكل التالي (من اليمين الى اليسار):



الشكل (4.3.9) توضيح التكرار في المخطط الهيكلي

لاحظ في هذا الشكل العمليات التالية:

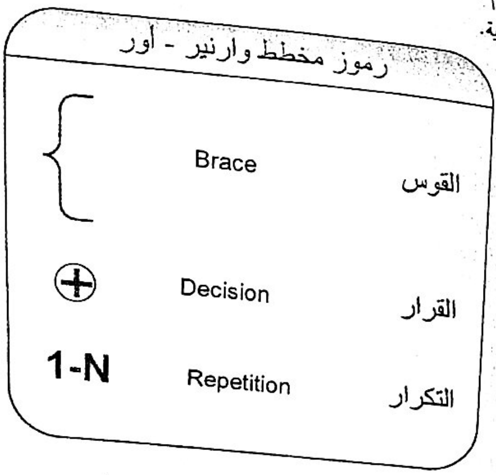
- اطبع كل معلومات الطالب — Print ALL student Information
- اقرأ سجل الطالب — Read student record
- نسّق معلومات الطالب — format student information
- اطبع معلومات الطالب — Print student information

والآن نبين مخطط وارنير ـ أور وهو مخطط أخر لتوضيح التصميم المعماري.

4.4 مخطط وارنير- أور Warnier–Orr Diagram

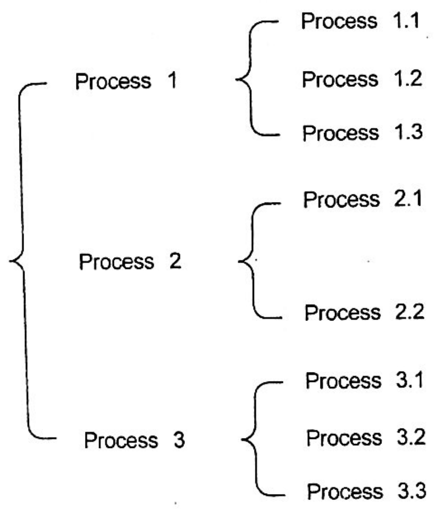
يعتبر مخطط Warnier–Orr Diagram أو بإختصار W\O أداة تصميم هرمية تستعمل لتجزئة العمليات الرئيسية Processes إلى عمليات فرعية Sub-processes.

ويستخدم هذا المخطط طريقة من فوق-إلى-أسفل (الهرمي)، ويوضح المنظومة من خلال تركيبة شجرية.



الشكل (4.4.1) رموز مخطط وارنير- أور

يبين الشكل 4.4.2 مثالا لنظام يتكون من ثلاث عمليات رئيسية (Processes)، حيث أن (العملية 1) (Process 1) تتكون من ثلاث عمليات فرعية، و(العملية 2) تتكون من عمليتين فرعيتين، والعملية 3 تتكون من ثلاث عمليات فرعية كما في الشكل التالي (من اليسار الى اليمين) :



الشكل (4.4.2) مخطط W\O

نلاحظ أن مخطط وارنيير - اور يعمل من فوق إلى أسفل، ومن اليسار إلى اليمين، ويستعمل الأقواس كرمز رئيسي..

وللمخطط تركيبات للتسلسل والقرار والتكرار.

مثال:

ارسم مخطط W/O لمنظومة مرتبات Payroll-System علما بأن:

صافي المرتب = المرتب الإجمالي - الخصميات + أجرة العمل الإضافي

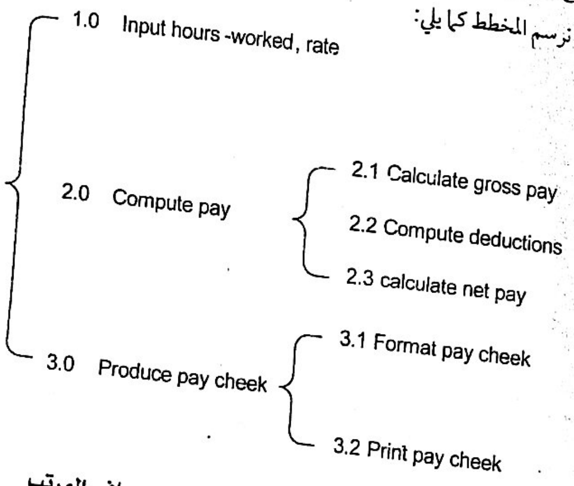
الحل:

أولا: نوضح عمليات المنظومة وهي :

ادخل ساعات العمل وأجرة الساعة الواحدة	Input hours-worked، rate

احسب المرتب	Compute pay
احسب المرتب الإجمالي	Compute gross pay
احسب الخصميات	Compute deductions
احسب المرتب الصافي	Compute net pay
استخرج صك المرتب	Produce paycheck
نسق بيانات الصك	Format paycheck
اطبع الصك	Print paycheck

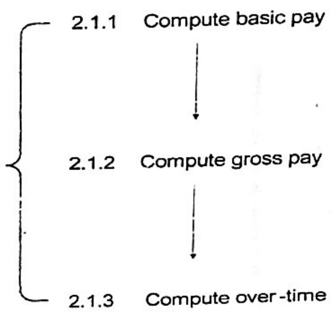
ثم نرسم المخطط كما يلي:



الشكل (4.4.3) مخطط W/ O لحساب صافي المرتب

ملاحظة هامة: لتوضيح ولتأكيد عملية التسلسل يمكننا استخدام السهم للإشارة إلى التتابع كما يلي:

احسب المرتب الأساسي	Compute basic pay
احسب المرتب الإجمالي	Compute gross pay
احسب قيمة شغل الوقت الإضافي	Compute over-time pay



الشكل (4.4.4) عملية التسلسل في مخطط W\O

اذكر مزايا الشكل W/O

ميزات مخطط W/O:

1. لا توجد صعوبة في رسم هذا المخطط.
2. لا يدخل هذا المخطط مباشرة في التفاصيل.
3. يمكن استخدامه مع شبيهة الشفرة والمخططات الانسيابية الهيكلية في التصميم التفصيلي (كتابة الخوارزمية).

4.5 مفاهيم وطرق التصميم Design Concepts and Techniques

يعتبر التصميم علما وفنا science and art ويستعمل كجسر يربط بين المتطلبات والبرمجة.

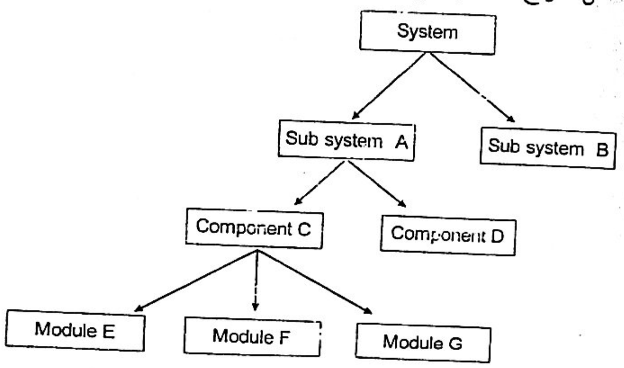
وفي هذا البند سندرس بعض المفاهيم والطرق التي يجب أن تؤخذ في الاعتبار عند تنفيذ عملية التصميم.

التنقيح خطوة خطوة Step–wise Refinement

عرف

التنقيح بالخطوة هو طريقة تصميم حيث يتم تقسيم النظام إلى نظم فرعية صغيرة، ثم تقسم هذه النظم الفرعية إلى أجزاء أصغر، وهكذا حتى تصبح الأجزاء عبارة عن وحدات بسيطة لا يمكن تقسيمها أكثر (انظر الشكل 4.5.1).

وبمجرد أن تنتهي عملية التقسيم يتم تخصيص المبرمجين لهذه الأجزاء لإنجازها كفريق عمل في شكل مشروع. والشكل التالي يبين تجزئة النظام إلى وحدات.



الشكل (4.5.1) تجزئة النظام

> **ملاحظة: من المفضل ألا يزيد طول الجزء البرمجي عن 40 سطرا أي في حجم عدد أسطر صفحة من كراسة ملاحظات.**

إخفاء المعلومات Information Hiding

يعتبر مفهوم إخفاء المعلومات Information Hiding مفهوما مهما حيث يتم إخفاء أسرار (منطق) جزء برمجي على بقية الأجزاء البرمجية الأخرى، وبتعبير آخر لا يتم تبادل المعلومات بين الأجزاء إلا عند الحاجة (مثلا عن طريق استعمال البارومترات).

عرف

التجزئة Modularity

الغرض من التجزئة Modularity هو إعداد منظومة سهلة من حيث التعامل.

يتحقق هذا الهدف بتقسيم المنظومة البرمجية المعقدة إلى أجزاء منفصلة بسيطة.

وتجمع هذه الأجزاء بعد ذلك في منظومة واحدة تحقق متطلبات المسألة المراد حلها.

ويجب أن نلاحظ هنا أن الجزء البرمجي Module هو عبارة عن مجموعة جمل برنامج تحتوي على:

- Input إدخال
- Output إخراج
- Function وظيفة
- Internal data بيانات داخلية
- Logic منطق

في داخل الجزء البرمجي يوجد:

- Logic منطق: الشفرة الإجرائية التي تحقق الوظيفة.
- Interfaces البيانات الداخلية

وفي خارج الجزء البرمجي يوجد:

- اسم الجزء البرمجي
- المدخلات: ما يدخله من بيانات
- المخرجات: ما يقوم بترجيعه
- الوظيفة: ما يفعله الجزء البرمجي لتنتج المخرجات.

ما هو المخطط الهيكلي وماذا يبين؟

التصميم الهيكلي Structured Design

التصميم الهيكلي Structured Design هو طريقة تقسيم المنظومة إلى أجزاء في شكل هرمي (شجري).

وتستعمل هذه الطريقة أدوات مثل المخطط الهيكلي ومخطط انسياب البيانات الهيكلي.

ويجب أن نأخذ في الاعتبار عند تقسيم المنظومة أن الأجزاء ذات المستوى المنخفض تقوم بالمعالجة، بينما تقوم الأجزاء ذات المستوى العالي بوظائف التحكم.

مزايا التصميم الهيكلي:

1. يؤدي إلى إعداد منظومة صحيحة ومفهومة وقابلة للصيانة.
2. يؤدي إلى إعداد منظومة واضحة قابلة للمتابعة ويمكن إعدادها من قبل فريق عمل.
3. يؤدي إلى إعداد منظومة سهلة الاختبار والتصحيح.
4. يؤدي إلى إعداد منظومة سهلة التجميع والتكامل.

4.6 معايير التصميم الجيد Criteria of Good Design

يقاس التصميم الجيد بمعايير أهمها الترابط والتجانس.

4.6.1 الترابط (التمرير) Coupling

الترابط هو مقياس للعلاقة بين الأجزاء البرمجية في التركيبة الهيكلية للمنظومة أي تمرير البيانات بين جزء برمجي إلى آخر.

وعند تصميم البرمجيات نتطلع دائما لأن يكون الترابط بين الأجزاء بسيطا وبأقل تمرير بيانات ممكن.

أنواع الترابط :

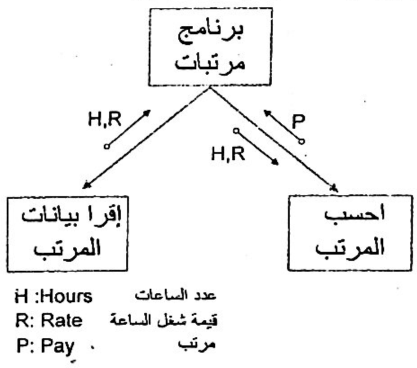
يمكننا أن نرتب أنواع الترابط بين جزء برمجي وجزء برمجي آخر إبتدأ من الأفضل وانتهاء إلى الأسوأ ويقاس بمقدار كمية البيانات التي يتم تمريرها بين الجزئين البرمجين كما يلي:

Data coupling	تمرير البيانات	Best	الأفضل
Stamp coupling	تمرير الطابع		
Control coupling	تمرير التحكم		
Common coupling	التمرير الشائع		
Content coupling	تمرير المحتويات	Worst	الأسوأ

1- تمرير البيانات Data Coupling

في هذا النوع يتم تمرير البيانات من جزء برمجي إلى آخر في شكل بارامترات.

مثال: يبين النوع تمرير البيانات (من اليسار الى اليمين):

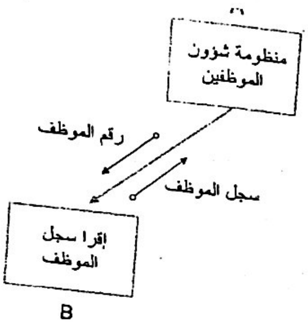


الشكل (4.6.1) تمرير البيانات Data coupling

يبين الشكل (4.6.1) تمرير البيانات حيث نلاحظ أن الجزء «احسب المرتب» تابع للجزء أو السواقة «برنامج مرتبات» وأنه يمكن الوصول إليه عن طريق قائمة المتغيرات البسيطة (البارامترات).

2 - تمرير الطابع Stamp Coupling

هنا نجد أن الجزء البرمجي يمرر شطرا من تركيبة بيانات (مثلا سجل) إلى جزء آخر.



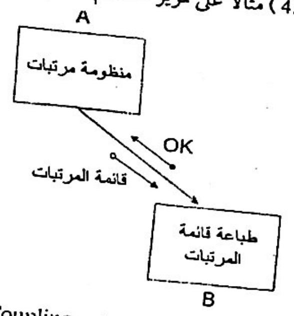
الشكل (4.6.2) تمرير الطابع Stamp Coupling

يبين الشكل (4.6.2) مثالا لتمرير الطابع ونجد هنا أن الجزء B تابع للجزء A ويتم التواصل عن طريق سجل.

3- تمرير التحكم Control coupling

في هذا النوع يتم تمرير راية تحكم Flag من جزء برمجي إلى آخر.

حيث يبين الشكل (4.6.3) مثالا على تمرير التحكم حسب الآتي:



الشكل (4.6.3) تمرير التحكم Control Coupling

هنا نلاحظ أن الجزء A يرسل قائمة المرتبات ويقوم الجزء البرمجي B بطباعة القائمة ثم يرسل راية تحكم (OK) للجزء A مؤكدا ان الطباعة قد تمت بنجاح .

4 - التمرير العام Common Coupling

تستعمل في هذا النوع الأجزاء البرمجية ما يعرف بالمتغيرات العامة التي تتعرف عليها كل الأجزاء البرمجية الرئيسية والفرعية.

وبتعبير آخر: جميع تراكيب البيانات (مثلا جميع السجلات أي ملف File) تكون مشتركة بين جميع الأجزاء البرمجية.

5 - تمرير المحتويات Content coupling

في هذا النوع من التمرير يتفرع أو يشير الجزء البرمجي إلى وسط جزء برمجي آخر Branch inside other module .

ويحدث هذا عادة عندما يستعمل جزء برمجي بيانات أو معلومات تحكم تقع داخل حدود جزء برمجي آخر. ويعتبر هذا النوع من التمرير الأسوأ ويجب تجنب استعماله.

4.6.2 التجانس أو الالتصاق Cohesion

التجانس أو الالتصاق Cohesion هو مقياس لقوة العلاقة بين جمل الجزء البرمجي الواحد.

والجزء البرمجي المتجانس هو ذلك الجزء الذي يقوم بمهمة واحدة أو وظيفة واحدة Single Function .

ويجب أن نستعمل دائما مستوى عالي من التجانس او الالتصاق بين جمل الجزء البرمجي الواحد.

أنواع التجانس:

نورد هنا قائمة بأنواع التجانس التي تستخدم عادة في كتابة الأجزاء البرمجية.

وهذه القائمة تبدأ من أسوأ نوع ألا وهو الالتصاق بالصدفة وتنتهي بأفضل نوع وهو الالتصاق الوظيفي (وظيفة واحدة فقط).

وضح مفهوم التجانس بالصدفة

1. التجانس بالصدفة Coincidental Cohesion

في هذا النوع لا توجد أي علاقة بين جمل الجزء البرمجي إلا علاقة الصدفة.

ويمكننا تصور هذا النوع عند تقسيم برنامج ذي 1000 جملة إلى أجزاء برمجية متساوية، بحيث يحتوي كل جزء برمجي على 50 جملة.

2. التجانس المنطقي Logical Cohesion

مكونات هذا الجزء البرمجي ترتبط بعلاقة من نوع ما (أي أنهم جميعا ينتمون إلى فئة ما).

مثال: الجزء البرمجي: «عدّل جميع المعاملات».

يتناول جميع المعاملات في دفعة واحدة، وليس معاملة واحدة في المرة الواحدة وكل معاملة تتشابه أجزاؤها مع المعاملة الأخرى.

3. الترابط الزمني Temporal Cohesion

تنفذ مكونات هذا الجزء البرمجي في نفس الفترة الزمنية.

وكمثال على هذا النوع من التجانس هو (جزء برمجي البداية Initiation module). حيث يتم تخصيص جزء برمجي ليتم إعطاء كل القيم الابتدائية للبيانات الخاصة بجميع الأجزاء البرمجية للمنظومة دفعة واحدة.

4. التجانس الاتصالي Communication Cohesion

العلاقة بين العناصر (جميع عناصر الأجزاء الفرعية للجزء البرمجي) في الجزء البرمجي تكمن في أنها تستعمل نفس فئة البيانات.

مثال:

الجزء البرمجي: (اعرض البيانات واطبع البيانات display data and print data) هو من النوع الاتصالي حيث يوجد دالتين فرعيتين في هذا الجزء هما الدالة اعرض والدالة أطبع وكلاهما يستعمل نفس البيانات.

5. التجانس التتابعي Sequential Cohesion

في هذا النوع تكون مخرجات دالة فرعية مدخلات دالة فرعية أخرى.

مثال:

الجزء البرمجي (احسب واطبع الحل compute and print solution) هو من النوع التتابعي حيث يتم حساب الحل أولا بالدالة الفرعية الأولى compute، ثم يتم طباعته بالدالة الفرعية الثانية Print. وهذا الترابط لايعتبر سيئا ولكن هناك ترابط أفضل منه وهو الترابط الوظيفي.

6. الترابط الوظيفي Functional Cohesion

هذا هو النوع المفضل حيث جميع العناصر في الجزء البرمجي تقوم بوظيفة واحدة.

مثال:

الجزء البرمجي (احسب المرتب compute pay) هو من النوع الوظيفي لان جميع عناصر هذا الجزء البرمجي تربطها علاقة الوظيفة (حساب المرتب).

4.6.3 التجزئة الرأسية والأفقية للأجزاء البرمجية

عند رسم المخطط الهيكلي في التصميم المعماري يجب أن نهتم بعدد الأجزاء البرمجية في كل مستوى وكم مستوى أو عمق مسموح به.

وهذا الموضوع متعلق بما يسمى المروحة المفتوحة والمغلقة:

المروحة المقفلة (Fan-in): تشير إلى عدد الأجزاء البرمجية التي تتحكم مباشرة في جزء برمجي معين.

المروحة المفتوحة (Fan-out): هي مقياس لعدد الأجزاء البرمجية التي هي تحت تحكم جزء برمجي معين.

والعدد الصحيح يتروح مابين 3 إلى 7 أجزاء برمجية تحت تحكم جزء برمجي معين وهو المروحة المفتوحة الجيدة.

وعدد المستويات يجب ألا تزيد عن 5 مستويات وهي المروحة المقفلة الجيدة.

4.7 أسئلة للمراجعة

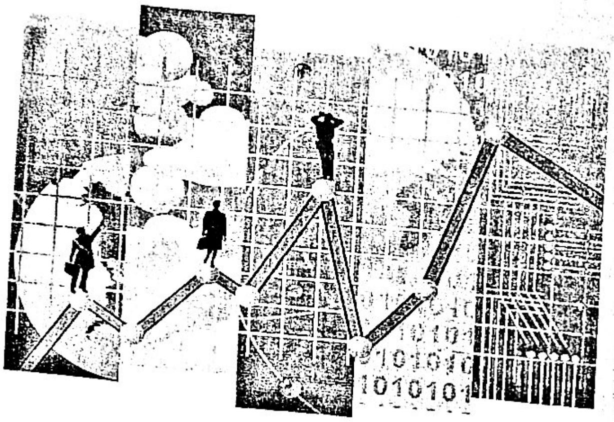
1. ما هي المكونات الرئيسية لوثيقة تحديد التصميم؟
2. عرّف ما يلي
   - structured design — التصميم المركب
   - modularity — التجزئة
   - step-wise refinement — التنقيح بالخطوة
   - structured programming — البرمجة الهيكلية
3. ما هو المخطط الهيكلي؟ وماذا يبين؟
4. صف بعض القواعد المهمة التي يجب إتباعها عند رسم المخطط الهيكلي ؟
5. ارسم مخططًا هيكليًا لنظام مصرفي مبسط يحتوي على الآتي :
   - Check for a balance — التأكد من الرصيد
   - Deposit a check — إيداع صك
   - Deposit cash — إيداع مبلغ مالي
   - Withdraw using a check — السحب باستعمال صك
   - Print a balance statement — طباعة الرصيد
6. المطلوب إعداد مخطط هيكلي لنظام التحكم في المخزون يقوم بالوظائف التالية :
   - Add an item — إضافة صنف
   - Update an item — تعديل بيانات صنف
   - Delete an item — إلغاء صنف
   - Inquire about an item — الاستفسار عن صنف

7. قارن بين منظومة المعاملة ومنظومة التحويل؟ وضح بمثال؟
8. اذكر بعض المزايا لشكل W\O؟
9. ما لفرق بين الترابط أوالتجانس؟
10. وضح مفهوم اقتران البيانات باستخدام شكل مناسب.
11. وضح مفهوم التجانس بالصدفة مستخدما مثالا مناسبا؟
12. مامعنى المروحة المفتوحة والمروحة المقفلة؟

الفصل الخامس

التصميم الإجرائي وتصميم البيانات

Procedural & Data Design



5.1 مقدمة

يتم تجهيز التصميم الإجرائي Procedural design عادة بعد تجهيز التصميم المعماري وتصميم واجهة المستخدم.

ويتم تجهيز مواصفات الإجراء لكل جزء برمجي تمّ ذكره وتعريفه في التصميم المعماري بشيء من التفصيل.

وتستخدم أدوات لتمثيل هذه المواصفات والخوارزميات باستخدام إمّا أدوات نصية (مثل شبه الشفرة Psuedocode) أو باستخدام المخططات (مثل مخطط الانسياب الهيكلي Structured flow chart).

ويجب ملاحظة أن أي تطبيق يمكن أن يتم تصميمه إجرائيا باستخدام أحد التركيبات الهيكلية المتعارف عليها في البرمجة الهيكلية.

وهذه التركيبات هي: التركيبة التسلسلية Sequence والتركيبة الشرطية Conditional والتركيبة التكرارية أو ما يعرف بتركيبة الحلقة أو الدوران Repetition .

في البنود القادمة سنشرح بعض هذه الأدوات.

5.2 المخطط الانسيابي الهيكلي Structred Flow Chart

بالرغم من أن مفهوم استخدام الصورة بصفة عامة أفضل من النص لتوضيح أو تصميم أي شيء، إلاّ أن هذا المفهوم لايؤخذ على عمومه خصوصا في مجال البرمجة وإعداد المنظومات، وخصوصا إذا كان هذا الرسم أو المخطط لا يفي بالمتطلبات أو يقود إلى تصميم غير جيد، الأمر الذي سيقود بدوره إلى منظومة متدنية الكفاءة.

قد كان المخطط الانسيابي Flowchart في السابق الأداة الأكثر شهرة بين المبرمجين في الستينيات وحتى السبعينيات لتمثيل التصميم الإجرائي، ولكن لسوء الحظ ونظرا لتعدد المتطلبات وكبر

حجم المنظومات لم يستطع المخطط الإنسيابي أن يجاري هذا التغيير وبالتالي قادنا إلى برامج ومنظومات صعبة الفهم والصيانة.

وأصبحت الشفرة المنبثقة من المخطط الانسيابي تسمى شفرة سباقيتي Spaghetti Code (تشبيها لها بالأكلة الإيطالية المعروفة بتشابكها) وهي شفرة يكثر فيها استعمال جملة Go to وهي الجملة التي قد يؤدي استعمالها إلى صعوبة في وضوح وتتبع البرنامج.

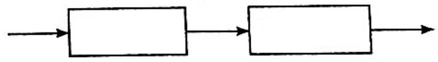
إن المخطط الانسيابي القديم (Flowchart) يمكن أن يستخدم في تصميم البرامج الصغيرة أو شرح مفاهيم التصميم المبسط لتعليم الطلبة بعض مفاهيم تصميم البرامج، ولكنه بالتأكيد غير مناسب لتصميم المنظومات المعقدة والكبيرة.

والسؤال الذي يطرح نفسه هنا هو ماهو الحل البديل؟

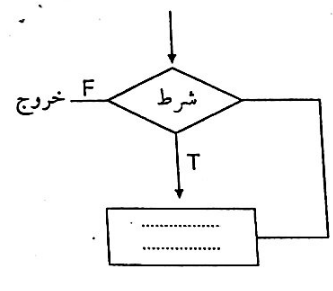
إن الحل البديل لهذه المشكلة يكمن في تقسيم البرنامج الكبير أو المنظومة إلى أجزاء برمجية صغيرة ثمّ استخدام ما يعرف بالمخطط الانسيابي الهيكلي لكل جزء.

وهذا يقودنا لتعريف المخطط الانسيابي الهيكلي Structured flowchart، وهو عبارة عن أداة تصميم إجرائي (تطوير للمخطط الانسيابي القديم) مكون من أشكال هندسية تمثل التركيبات الثلاث (بنيات) الأساسية لأي برنامج وهو مأخوذ من تركيبات البرمجة الهيكلية والذي يتم توضيحه كالآتي:

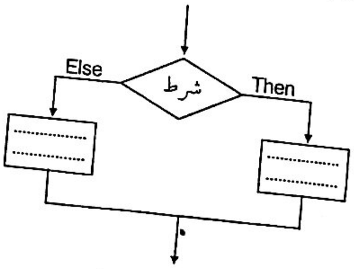
a) البنية التتابعية Sequential Construct



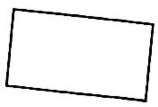
b) بنية نفذ- طالما While-do Construct



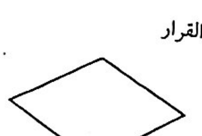
c) بنية إذا- وإلا If -Then-Else Construct



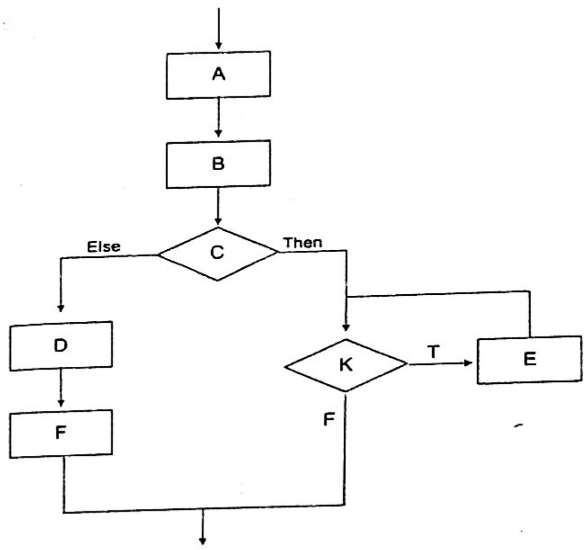
لاحظ أن المستطيل يستعمل للمعالجة



و يستعمل الشكل التالي لاتخاذ القرار



و الاختلاف بين المخطط الانسيابي القديم والمخطط الانسيابي الهيكلي هو أن المخططات الانسيابية الهيكلية تعتمد على استعمال مصطلحات البرمجة الهيكلية والتي منها if-then-else و do-while ويبين الشكل (5.2.1) مثالا لمخطط انسيابي هيكلي، لاحظ في الشكل أن الرموز A، B, D, F, E تعبّر عن جمل مثل جملة التعيين أو الادخال أو الإخراج، ولكن الرمزين C,K هما عبارتان منطقيتان قيمتهما إما صواب أو خطأ.



الشكل (5.2.1) مخطط انسيابي هيكلي

لكي ننفذ هذا المخطط الانسيابي الهيكلي نقوم بترجمته إلى الشفرة التالية:

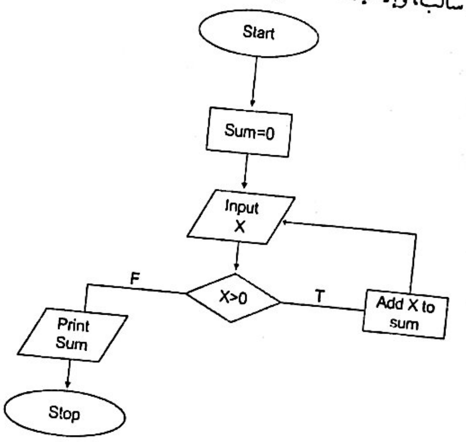
```
Do      A
Do      B
IF      C       Then
                While (K)
                Do      E
                Loop
ELSE
        Do D
        Do F
End if
```


مثال 1:

ارسم مخطط انسيابي هيكلي لبرنامج يقرأ أرقام موجبة ويطبع المجموع.

الحل :

في هذا المثال نوضح فكرة الحلقة، حيث تقوم الحلقة بقراءة عدد وإضافته إلى المجموع طالما أن هذا العدد غير سالب، وإلاّ فإن الحلقة تتوقف.



الشكل (5.2.2) مخطط انسيابي هيكلي لإيجاد المجموع

مثال 2:

تقوم شركة كهرباء بتسعير الاستهلاك كما يلي:

- 2 درهم على الكيلووات الواحد إذا كان الاستهلاك أقل من 1000 كيلووات.
- 5 دراهم على الكيلووات الواحد إذا كان الاستهلاك 1000 كيلووات فما فوق.

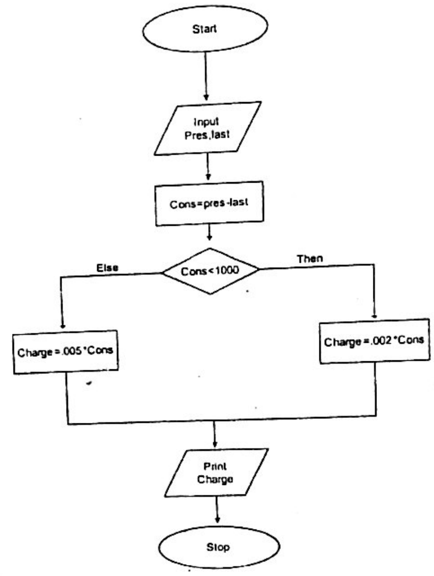
ارسم مخططا انسيابيا لطباعة فاتورة بحيث تبين الفاتورة القراءة السابقة والحالية

الحل:

نستخدم الرموز الآتية:

القراءة الحالية	Pres
القراءة السابقة	Last
الاستهلاك	Cons
القيمة المطلوبة (الفاتورة)	Bill or Charge

فيكون المخطط الانسيابي الهيكلي كما في الشكل (5.2.3):



الشكل (5.2.3) مخطط انسيابي هيكلي لحساب تكاليف الكهرباء

5.3 شبه الشفرة Pseudocode

نستعرض هنا أداة أخرى لتمثيل التصميم الإجرائي بطريقة نصية بدلا من استخدام المخطط.

وتسمى هذه الأداة بشبه الشفرة Psuedocode لأنها تشبه إلى حد بعيد شفرة لغة برمجة مثل باسكال.

شبه الشفرة هي أداة تصميم إجرائي نصي تبين منطق البرنامج (خطوات الخوارزمية) بلغة دقيقة ومفهومة مستخدمة تركيبات البرمجة الهيكلية الثلاثة: التسلسلية والشرطية والتكرارية.

والمثال التالي يستخدم شبه الشفرة كأداة تصميم قبل ترجمتها إلى برنامج:

مثال 3:

اكتب شبه الشفرة Pseudocode لبرنامج يحسب فاتورة الكهرباء (charge) باستخدام القراءة السابقة للعداد (last) والقراءة الحالية (Pres) حسب الشروط الآتية:

إذا كانت كمية الاستهلاك بالكيلوات-ساعة (cons) اكبر من 1000 يكون سعر الكيلووات 50 درهم، وإذا كانت أقل أو يساوي 1000 فيكون السعر 20 درهم لكيلوات.

الحل:

```
Read pres، last
Cons = pres –last
If cons > 1000 then
    Charge= (50*cons) / 1000
Else
    Charge=(20*cons) / 1000
End if
Print pres، last، cons، charge
End
```

ملاحظة: نقسم الناتج على 1000 بغرض التحويل من درهم إلى دينار

مثال 4:

اكتب شبه الشفرةPseudocode لحساب مجموع مبيعات محل اثاث، علما بأن عدد القيم للمبيعات مجهول ولكن توجد قيمة سالبة في النهاية كمؤشر لنهاية البيانات.

الحل: نقوم أولا بتحديد المتغيرات كالتالي:

قيمة المبيعات	X
المجموع:	S

وبذلك تكون شبه الشفرة المطلوبة كما يلي:

```
S = 0
Input X
While X > 0
    S = S + X
    Input X
End while
Print S
End
```

5.4 أداة هرمية الإدخال والعمليات والإخراج

Hierarichal Input Process Output (HIPO)

تستخدم أداة هيبو HIPO التي تم إعدادها من قبل شركة IBM في التصميم والتوثيق.

وقد ظهر هذا المخطط في بداية السبعينيات لمساعدة مديري المشاريع للمشاركة في مرحلة التصميم.

وتتكون هذه الأداة من مجموعة من المخططات وذلك من أجل تسهيل التصميم بطريقة من - فوق- إلى - أسفل. ولكن هذه الأداة ليست بديلا عن شبه الشفرة أو المخطط الإنسيابي الهيكلي وفي الواقع تعتبر مكملة لها.

كيف نستعمل مخطط HIPO هيبو؟

يستخدم مخطط HIPO في تقسيم المنظومة إلى عدة أجزاء بحيث أن كل جزء يتم تعريفه من خلال مدخلاته وعملياته ومخرجاته.

مخططات HIPO توضح هرمية وتنظيم الأجزاء وعلاقتها ببعضها البعض.

وهي كذلك تستخدم شبه الشفرة لتوضيح منطق كل جزء.

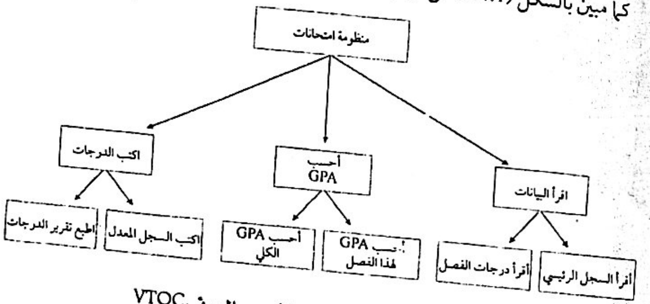
مخططات هيبو: HIPO Diagrams

اذكر الاشكال الثلاثة المستخدمة عادة في مخطط HIPO

تتكون رزمة مخططات HIPO من ثلاثة أنواع من المخططات هي:

مخطط الفهرس المرئي Visual Table of Contents Diagram (VTOC)

وهو عبارة عن مخطط هرمي يبين أجزاء المنظومة ووظائفها بأسلوب عام كما مبين بالشكل (5.4.1) من اليمين إلى اليسار:



الشكل (5.4.1) مخطط الفهرس المرئيVTOC

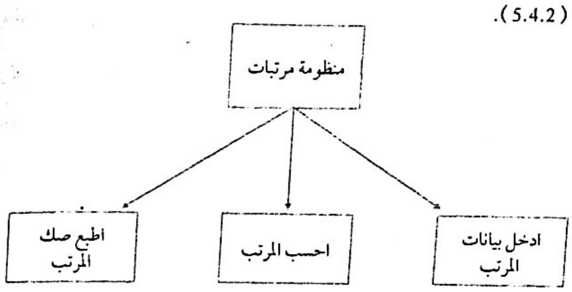
في هذا الشكل استخدمنا GPA وهي إختصار Grade Point Average وتعني متوسط الدرجات للطالب في فصل دراسي معين أو لكل الفصول.

> **ملاحظة: لاحظ في الشكل 5.4.1 أن المخطط يتكون من مستويين، المستوى الأول وهو بدوره يتكون من ثلاثة أجزاء عامة، والمستوى الثاني يتكون من أجزاء أبناء المستوى الأول. وفي كل مستوى نلاحظ أن الأجزاء يتم تنفيذها من اليمين إلى اليسار.**

مخطط الشكل العام Overview Diagram

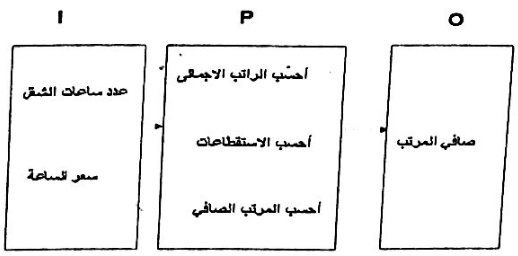
يبين هذا المخطط تفصيلا أكثر حول الأجزاء (الوظائف) الرئيسية للمنظومة التي تم تحديدها في مخطط VTOC. وهو يحلل أجزاء (أي وظائف) النظام إلى فروع باستخدام مخطط I-P-O.

ولعرض مثال عن مخطط الشكل العام، ندرس نظام رواتب لموظفي مؤسسة ما كما هو مبين بالشكل (5.4.2).



الشكل (5.4.2) مخطط VTOC لمنظومة مرتبات

نستطيع تكوين مخطط الشكل العام للجزء الخاص: «حساب المرتب» من مخطط الفهرس المرئي كما هو مبين بالشكل (5.4.3).

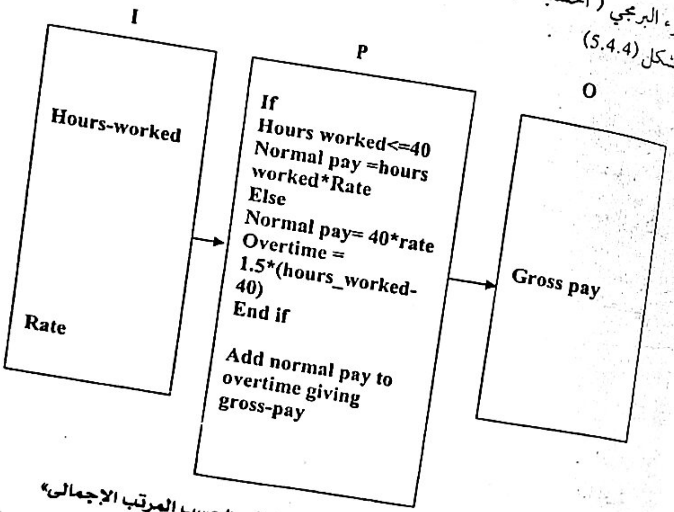


الشكل (5.4.3) مخطط الشكل العام

المخطط التفصيلي Detail Diagram

يبين هذا المخطط أكبر قدر ممكن من التفاصيل حول وظيفة معينة.

ويعتمد مقدار التفاصيل على مدى تعقد المسألة المطلوب حلها. ونستطيع مثلا توضيح الجزء البرمجي (احسب الراتب الإجمالي) المأخوذ من مخطط الشكل العام على النحو المبين بالشكل (5.4.4) .



الشكل (5.4.4) المخطط التفصيلي للجزء «احسب المرتب الاجمالى»

ملاحظات :

1. بالنسبة للأنظمة الصغيرة، قد نحتاج فقط لمخطط أو اثنين، ولكن بالنسبة للأنظمة الكبيرة نحتاج إلى الأنواع الثلاثة المذكورة أعلاه.
2. نستطيع أن نستعمل شبه الشفرة مع مخططات هيبوHIPO الهرمية إذا لزم الأمر.
3. من عيوب مخطط HIPO التفصيلي أن الصندوق المخصص للمعالجة قد لا يكون كافيا أحيانا.

مزايا مخططات HIPO :

من مزاياها أنها:

1. يمكن اكتشاف وتصحيح الأخطاء في التصميم باستعمال المخططات الثلاثة: مخطط الفهرس المرئيVTOC ومخطط الشكل العام والمخطط التفصيلي.
2. تعتبر مخططات HIPO أداة توثيق ممتازة عند كتابة مواصفات المتطلبات ومواصفات التصميم ودليل المستخدم ...الخ.
3. تعتبر أداة جيدة للتواصل بين الزبون ومعد البرمجيات والمدراء...الخ. فعلى سبيل المثال يمكن إعداد التوثيق من مخطط VOTC العام ثم الدخول في التفاصيل في المخطط التفصيلي.
4. يعطى التوثيق لأفراد من مختلف المستويات الفنية. فمثلا قد لا يفهم المستخدم (الزبون) إلا المستوى الأول، أي مخطط VOTC.

بعد أن استعرضنا التصميم التفصيلي أو الإجرائي نقوم الآن بشرح نوع آخر من التصميم ويهتم بتصميم هيكلية البيانات الخاصة بالمنظومة.

5.5 تصميم قاعدة البيانات Data Base Design

يعتبر تصميم قاعدة البيانات Data Base Design من النشاطات المهمة عند إعداد المنظومات. ويبدأ تصميم البيانات من مرحلة التحليل حيث يتم التعرف على البيانات الخاصة بالمنظومة ومن ثمّ تنظيمها وهيكلتها حتى تصبح سهلة التخزين والاسترجاع والتغيير.

والهدف الأساسي من تصميم البيانات هو تحويل البيانات التي تمّ إعدادها في مرحلة التحليل إلى هياكل بيانات ليتم استخدامها في مرحلة البرمجة.

وعند تصميم البيانات يجب تجهيز الآتي:

- قاموس البيانات Data Dictioary للتعرف على كل عناصر البيانات الخاصة بالمنظومة وخصائصها.
- الكائنات والعلاقة بين هذه الكائنات.

- الخصائص الخاصة بكل كائن.
- مخطط الكائنات العلائقية ERD
- تصميم قاعدة بيانات علائقية المتحولة من مخطط الكائنات العلائقية.

في البنود التالية سنتعرض إلى مخطط الكائنات العلائقية وقاموس البيانات.

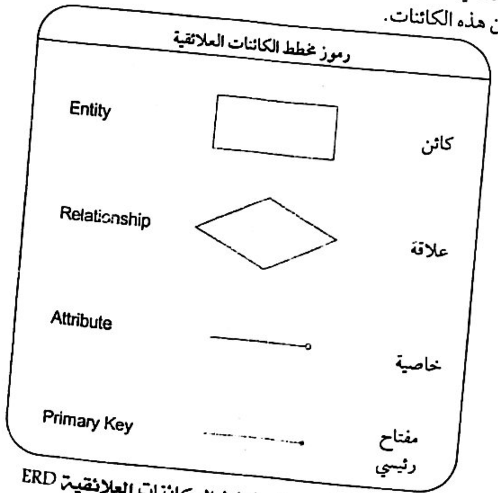
تكلم عن مخطط الكائنات العلائقية

5.6 مخطط الكائنات العلائقية ERD

يعتبر بيتر تشن هو أول من أعدّ أداة التصميم Entity Relationship Diagram ERD - وكان ذلك في أواخر السبعينيات لغرض تصميم نظم قواعد البيانات العلائقية.

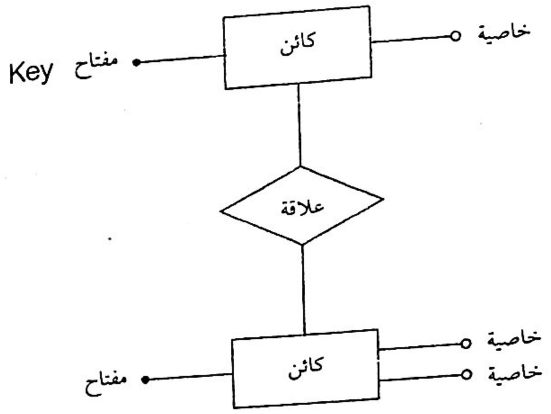
وأصبحت هذه الأداة ذات أهمية في تصميم هياكل قواعد البيانات.

والهدف الرئيسي من هذا المخطط هو توضيح عناصر البيانات (الكائنات) وخصائصها والعلاقات بين هذه الكائنات.



الشكل (5.6.1) رموز مخطط الكائنات العلائقية ERD

في مخطط الكائنات العلائقية يمكن أن يوجد كائنان مرتبطان بعلاقة مع خاصية ومفتاح رئيسي كما في الشكل (5.6.1).



الشكل (5.6.2) ERD مخطط الكائنات العلائقية

مثال:

في الكائن (Customer أي الزبون) يوجد أربعة خصائص (4 حقول) ومفتاح رئيسي واحد:

رقم الزبون (مفتاح)	Primary key: Customer-id
اسم الزبون (الحقل الأول)	Attribute 1: Customer-name
العنوان (الحقل الثاني)	Attribute 2: Address
الحالة {جديد / قديم} (الحقل الثالث)	Attribute 3: Status
الرصيد (الحقل الرابع)	Attribute 4: Balance

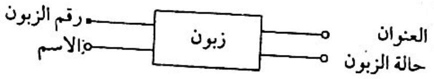
مثال:

يعتبر الكتاب والزبون كائنين

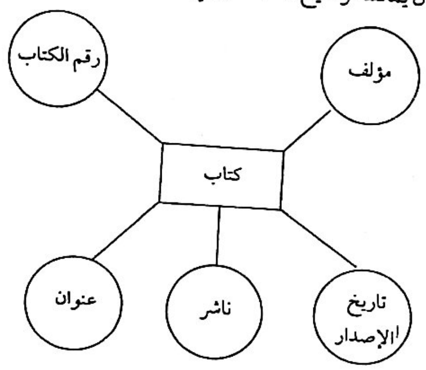
والعلاقة بينهما كما يلي:

- الزبون يستعير كتاب
- الزبون يستفسر عن كتابا
- الزبون يعيد كتابا

ويمكن تمثيل الخصائص الخاصة بكل كائن بإحدى الأشكال التالية:



على سبيل المثال يمكننا توضيح العلاقة كما يلي:





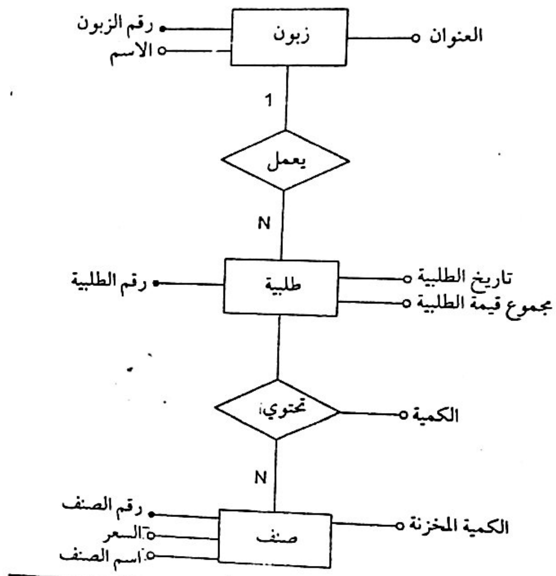
لاحظ أن رقم الكتاب هو المفتاح الرئيسي.

كما أن مخطط الكائنات قد ساعدنا في الحصول على أساس لتكوين جدوال قاعدة البيانات، حيث يمكن تكوين جدول لكل كائن.

مثال: أرسم مخطط كائنات علائقية لمنظومة مبيعات حيث يقوم الزبون بعمل طلبية تحتوي على أصناف متنوعة .

الحل:

المخطط التالي يبين بطريقة مبسطة الكائنات في هذه المنظومة والعلاقة بينها، مع توضيح مجموعة محددة من الخصائص حسب الشكل التالي:



الشكل(5.6.3) مخطط كائنات علائقية لمنظومة مبيعات

5.7 المصفوفة الكائنية Entity Matrix

في الأمثلة السابقة استخدمنا مخطط الكائنات العلائقية مباشرة ولكن من المفضل استخدام ما يسمى المصفوفة الكائنية في المنظومات المعقدة والكبيرة.

المصفوفة تتكون من صفوف وأعمدة، وكل خلية في هذه المصفوفة تحتوي على العلاقة بين كائنين بين الصف والعمود كما في المثال التالي :

مثال: بين المصفوفة الكائنية لمنظومة وضع إعلانات لبيع عقارات عن طريق الانترنت

ملاحظة:

الكائنات في هذه المنظومة:

- الزبون (متصفح الانترنت) Client
- الإعلان Advertisement
- العقار Real estate
- مالك العقار Owner

والعلاقات هي:

- المالك يضع عقارا Owner places Advertisement
- المالك يملك عقارا Owner has real estate
- الإعلان يخص عقارا Adver. Belongs to real estate
- الزبون يستعرض إعلان Client searches Adver

بهذا تكون المصفوفة الكائنية لهذه العلاقات كما يلي:

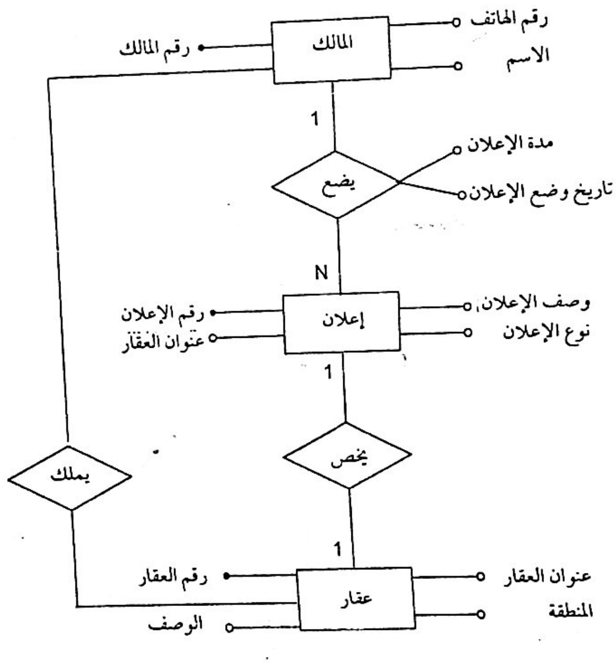
	المالك	إعلان	عقار	الزبون
المالك	X	يضع	يملك	X
إعلان	X	X	يخص	X
عقار	X	X	X	X
الزبون	X	يستعرض	X	X

ملاحظة:

المصفوفة تمثل منظومة إعلانات عقارات حيث أن المالك يدخل على الانترنت باستخدام كلمة عبور ورقم سري لوضع أو إدخال بيانات الإعلان حيث يتمكن أي زبون (شخص) من الدخول على هذا الموقع واستعراض العقارات المعروضة والبيانات الخاصة بها ثم في حالة الرغبة يمكن أخذ رقم الهاتف والاتصال مباشرة بالمالك لإتمام صفقة البيع.

وبعد تكوين هذه المصفوفة يصبح من اليسير تحويل هذه المصفوفة إلى مخطط كائنات علائقية

ونوضح مخطط الكائنات العلائقية كما يلي :



الشكل (5.7.1) مخطط كائنات علائقية لمنظومة عقارات

مثال:

المخطط التالي لشركة تعمير تعمل على مشاريع متعددة في مجال البناء حيث الكائن هنا هو (مشروع) له خصائصه كالتالي:

- رقم المشروع
- أسم المشروع
- الميزانية
- تاريخ البداية
- تاريخ النهاية
- نوع المشروع

والكائن (شركة) له الخصائص الاتية:

- اسم الشركة
- عنوان الشركة
- مجال عمل الشركة

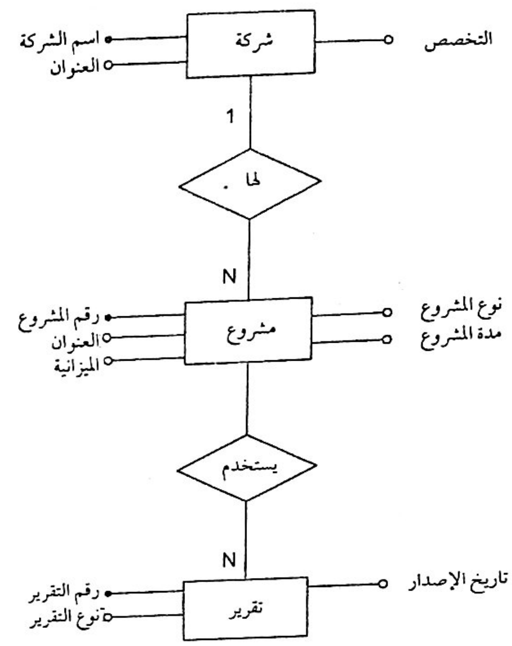
والكائن (تقرير) له الخصائص الاتية :

- رقم التقرير
- نوع التقرير
- تاريخ اصدار التقرير
- عدد اصدارات التقرير

العلاقات كالآتي:

- الشركة لها مشاريع (Company owns projects)
- المشروع يستخدم تقارير (Project uses report)

مخطط الكائنات العلائقي لهذه المنظومة كما يلي:



الشكل (5.7.2) مخطط كائنات علائقية لمنظومة مشاريع بناء

5.8 قاموس البيانات Data Dictionary

قاموس البيانات هو أداة تحليل وتصميم وهو مخزن معلومات مركزي خاص ببيانات المنظومة. وهذه البيانات تتمثل في عناصر البيانات أو الحقول.

ويمكن تعريف قاموس البيانات أيضا بأنه قائمة منظمة تحتوى على كل عناصر البيانات

الخاصة بالمنظومة ليتم استخدامها من قبل المبرمج ومحلل النظم والمستخدم لتكون أداة تفاهم على نوع وخصائص بيانات المدخلات والمخرجات لأنها الأساس لتصميم قاعدة البيانات.

ويجب أيضا تحديد وتفصيل خصائص كل عنصر بيانات والتي من أهمها:

1. اسم الحقل (عنصر البيانات)
2. الاسم المختصر لعنصر البيانات
3. وصف نصي لعنصر البيانات
4. نوع عنصر البيانات
5. طول عنصر البيانات
6. نطاق ومدى قيم عنصر البيانات
7. القيمة الافتراضية لعنصر البيانات
8. مكان تخزين عنصر البيانات
9. موقع استخدام عنصر البيانات

نكتفي بعرض مثال مبسط لبعض هذه الخصائص:

1. اسم العصر Item-no
2. مختصر العنصر I-no
3. وصف الحقل يستخدم كمفتاح رئيسي للوصول إلى سجل معين
4. نوع الحقل Integer
5. طول الحقل 4
6. نطاق الحقل 1 الى 9999

ويمكن تجهيز قاموس البيانات على شكل جدول أو تعبئة هذا القاموس التابع لنظم إدارة قواعد البيانات مثل Access أو SQL أوORACLE التي تقدم قاموس البيانات بطريقة آلية.

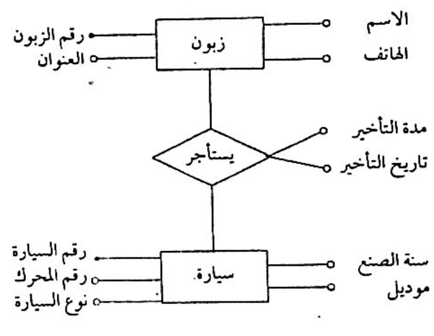
5.9 أسئلة للمراجعة

1. ارسم مخطط انسيابي هيكلي لحساب الضريبة y لدخل x بحيث:

$Y= 0.15*x$ When $x <200$

$Y=0.23*x$ When $x>200$ or $x=200$

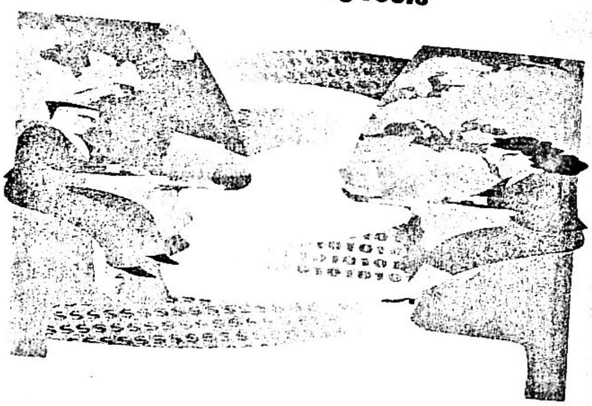
2. قارن بين المخطط الانسيابي التقليدي والمخطط الانسيابي الهيكلي ومتى يجب استخدام النوع الثاني؟
3. ارسم مخطط انسيابي هيكلي لحساب متوسط مجموعة من القيم الموجبة بحيث توضع في نهاية البيانات قيمة سالبة كمؤشر للنهاية.
4. حوّل المخطط الانسيابي الهيكلي في المسألة (3) إلى شبه الشفرة
5. اذكر الأشكال الثلاثة المستخدمة عادة في أداة التصميم HIPO.
6. لماذا تعتبر HIPO من أدوات التصميم الجيدة؟
7. تكلم بإيجاز عن مخطط الكائنات العلائقية ERD .
8. أرسم ERD لمنظومة مركز التحليل الطبي .
9. مالعلاقة بين الكائن (زبون) والكائن (سيارة) في الشكل التالي:



الفصل السادس

أدوات البرمجة

Programming Tools



6.1 مقدمة

لقد مرت خلال السنين الخمسين الماضية تغييرات جذرية على أدوات البرمجة الأساسية المتمثلة في لغات البرمجة.

ففي بداية الخمسينيات كان المبرمجون يكتبون شفرة البرنامج بلغة الآلة ولغة الأسمبلي .

ولكن بعد تعقد البرامج ونتيجة لزيادة الاحتياجات والمتطلبات، اتضح أن العديد من البرامج التي كتبت بهذه اللغات في ذلك العصر كانت صعبة القراءة والتعديل فظهرت لغات سهلة البرمجة مثل الفورتران للتطبيقات العلمية والكوبول للتطبيقات التجارية.

وفي السبعينيات حدث تطور آخر حيث تمّ تقديم مفهوم ما يعرف باللغات الإجرائية Prcocedural Programming وكانت بدايتها لغة باسكال Pascal ولغة C ولغة الغول Algol والتي تقسم البرنامج إلى مجموعة أجزاء برمجية تسمى (إجراءات) Procedures وتتم تسمية هذه الإجراءات بجمل تسلسلية Sequential Calls وينتهي البرنامج بمجرد الانتهاء من تنفيذ كل هذه الإجراءات.

وهذه الإجراءات سهلة الفهم والكشف عن أخطائها، إلاّ أنه من عيوبها أن البيانات Data والإجراءات Procedure منفصلان وأي تغيير بسيط في الإجراء يتطلب تغييرا في البرنامج الرئيسي الذي إستدعى هذا الإجراء وهذا بالطبع سيجعل من صيانة البرمجة مهمة صعبة.

ثمّ زاد بعد ذلك الاهتمام بشكل المخرجات والنتائج التي تظهر على الشاشة أو على الطابعة بما يعرف بواجهة المستخدم Graphical User Interface الرسومية لعرض وطباعة النتيجة بصورة جيدة، فاستخدمت في أواخر الثمانينيات اللغات المرئية Visual programming لهذا الغرض والتي من أشهرها لغة بيسك المرئي Visual basic .

وظهرت أيضا لغات قواعد البيانات Data Base Languages وتسمى أيضا نظم إدارة قاعدة البيانات DBMS مثل Oracle بعد ازدياد الطلب على منظومات لها أسبقية الاهتمام بالتعامل مع (النضائد) في التخزين والترجيع ومن أمثلة هذه المنظومات منظومة حفظ ملفات الموظفين والأرشيف الالكتروني ....الخ.

وفي التسعينيات ظهرت لغات البرمجة الشيئية Object-orieted Programming Languages من أجل الحصول على برمجيات ذات جودة عالية وجدول زمني أقل.

وفي هـذا البـاب سنتنـاول عـدة مصطلحـات مستخدمـة في البرمجـة والتي يجب على القارئ الاطـلاع عليهـا من بـاب الفائـدة، وسنتنـاول أيضا أنـواع طرق البرمجـة الخاصة بإعداد البرمجيات.

ما الفرق بين البرمجة الهيكلية والبرمجة الكائنية

6.2 البرمجة الهيكلية Structured Programming

لقد تمّ استعمال البرمجة الهيكلية Structured Programming لغرض حل مشكلة ما يعرف بشفرة السباقيتي Spagetti Code المتشابكة، والتي تنتج عادة من سوء استخدام جـملة GO TO حيث تسبب في إنتاج برامج صعبة التتبع والتعديل والصيانة.

أما البرمجة الهيكلية (ولها تسمية مشابهة وهي البرمجة الإجرائية) فتعتمد على أسلوب (من فوق إلى تحت) في تصميم البرنامج أي تجزئة البرنامج إلى وحدات تسمى إجراءات، واستخدام مجموعة تراكيب في البرنامج والتي منها تراكيب جملة إذا–وإلاّ IF-THEN-ELSE وجملة طالما while.

6.3 البرمجة بالحدث والمرئية Visual Programming & Event

تختلف لغات البرمجة بالحدث والمرئية Visual & Event Programming عن لغات البرمجة الهيكلية في طريقة التعامل مع انسياب البيانات حيث كان المستخدم لا يسيطر على تسلسل العمليات التي تحدث أثناء التنفيذ.

أما هذه اللغات فتضع المستخدم في موقع السيطرة على الأحداث فهو الذي يختار نوع المعاملة

من البرنامج (بالضغط على الفأرة أو الانتقاء من قائمة) أي أن البرنامج لايتم تنفيذه بالكامل إلاّ برغبة المستخدم. لهذا السبب تسمى: اللغات الموجّهة بالحدث Event-driven languages .

ويتم بدء الحدث event بواسطة المستخدم بالنقر على أيقونة أوالضغط على مفتاح مما يسبب في حدوث عملية Process لذلك فإن هذه اللغات تعمل في بيئة رسومية وليس بيئة نصية.

وتتمتع البرمجة المرئية بتوفر ما يعرف بصندوق الأدوات الذي يمكن المبرمج من إعداد واجهة رسومية مناسبة للمستخدم.

6.4 البرمجة الشيئية Object–Oriented Programming

لقد أصبحت البرمجة الشيئية Object–Oriented Programming (وتسمى أيضا البرمجة الكائنية) شائعة في التسعينيات، ولا زالت تنتشر كمفهوم مفضل للبرمجة خصوصا للشباب الذين لم يعاصروا البرمجة الهيكلية.

وهذا المفهوم نتاج طبيعي لما هو موجود في الحياة اليومية، ويستخدم مصطلحات بيئة الزبون.

إن مناداة مجموعة إجراءات تسلسلية ليست من سمات البرمجة الشيئية، ولكن يتم التفاعل بين مجموعة من الكائنات Objects للوصول إلى انسياب عام، حيث يستطيع المبرمج أن يقسم برنامجه على شكل كائنات أو أشياء قابلة لإعادة الاستعمال reuse.

ويتكون كل كائن من خصائص Attributes وعمليات Operations يتم تغليفهما Encapsulated داخل هذا الكائن.

ولنأخذ مثالا بسيطا يوضح ولو بشكل من التبسيط مفهوم الكائنات :

يعتبر الكوب كائن ملموس يمكن ملؤه أو تفريغه من السوائل (بالماء أو الشاي أو القهوة)، وهذا يعني أن هناك على الأقل عمليتين يمكن إجراؤهما على القطعة (الكوب) هما الملء والإفراغ.

كذلك للكوب خصائص مثل اللون والشكل والوزن والسعر وغيره.

ونأخذ مثالا آخر وهو كائن الفاتورة لها خصائص مثل رقم الفاتورة وتاريخ الفاتورة وقيمة الفاتورة وعمليات مثل إصدار الفاتورة وإلغاء الفاتورة.

وبهذا يمكن تعريف الكائنObject بأنه تركيبة برمجية تقوم بتغليف حالته (خصائصه) وتصرفاته (الأفعال أو الخدمات) التي يؤديها، والكائن هو حالة خاصة من الفصيلة Class (أي أن كل كائن ينتمي إلى فصيلة معينة).

ويتكون الكائن البرمجي من البيانات وطرق معالجة البيانات كالآتي:

- ■ كائن Object
- ■ بيانات (خصائص) Data
- ■ طرق وعمليات Methods

أسم الكائن
بيانات
طرق

الطريقة Method يمكن تعريفها بأنها الخوارزمية التي تؤثر على البيانات داخل الكائن، أمّا البيانات DATA فغالبا ما تكون مخفية ويتم تكوينها عند انشاء الكائن.

ولمزيد من التوضيح لمفهوم البرمجة الشيئية(الكائنية) يجب أن نفهم مصطلحاتها ومنها:

■ الفصيلة Class

الفصيلة عبارة عن حالة عامة من الكائن تخفي (تغلّف) البيانات والطرق (العمليات) الخاصة به.

■ الطريقة Method

الطريقة عبارة عن عملية أو معالجة تتم على بيانات الكائن.

■ فصيلة فرعية Subclass

عبارة عن عائلة متفرعة من الفصيلة الأم وترث عنها البيانات والطرق الخاصة بها.

■ الخصائص Attributes

صفات الكائن أو الفصيلة تسمى الخصائص.

وفي مثالنا عن الكائن: الكوب نلاحظ أنه يملك خصائص اللون والشكل والوزن ... الخ. وخاصية اللون تملك نطاقا من القيم (أسود، أزرق، أخضر، أحمر ...)

■ التصرف Behavior

هو عمل أو فعل يقوم به الكائن عند تمرير رسالة Message أو إجابة رسالة Response.

■ الرسالة Message

الكائنات تتفاهم فيما بينها عبر الرسائل. والرسالة لها مرسل ومستقبل.

■ التغليف Encapsulation

الكائن يحتوي على البيانات والعمليات في علبة مقفلة. وبهذه الطريقة نحقق مفهوم اخفاء المعلومات الخاصة بكائن ما على كائن أخر، وإذا حدث تبادل فلن يتم إلاّ عبر الرسائل.

■ الوراثة Inheritance

وتعني أن الفصيلة الفرعية N ترث كل الخصائص والطرق(العمليات) المتعلقة بالفصيلة الأم M، وبهذه الطريقة كل العمليات وتراكيب البيانات التي تم تصميمها وتنفيذها للفصيلة M تصبح متوفرة للفصيلة الفرعية N.

وبذلك يتحقق مفهوم إعادة الاستعمال Reuse مباشرة.

■ تعدد الأشكال Polymorphism

تعنى الكلمة باللاتيني تعدد الأشكال Many forms .وهي آلية تسمح لاسم معين بتمثيل عدة شفرات بمعنى كتابة شفرة مزاجية تتصرف بعدة أشكال.

ولتوضيح المعنى نأخذ المصطلح (اقفل) فيمكن أن يعني:

1. اقفل الباب.

2. اقفل النافذة.
3. اقفل الخزانة.
4. اقفل الحساب المصرفي.
5. اقفل الصندوق.

فالكلمات باب، صندوق، نافذة، حساب، خزانة هم كائنات Objects.

لهذا يمكن تطبيق هذا المصطلح على كل الكائنات، وكل كائن يفهم معنى كلمة اقفل بطريقته الخاصة. وفي كل حالة يمكن أن نقول اقفل فقط.

وبلغة البرمجة يعني السماح للفصيلة Class بتمثيل عدة شفرات يتم اختيارها بآلية معينة.

و باستخدام تعدد الأشكال يمكن للعديد من العمليات المختلفة أن يكون لها اسم واحد.

فمثلا قد يكون اسم الوحدة البرمجية هو (المساحة) حيث تقوم الوحدة البرمجية بحساب مساحة مجموعة من الأشكال الهندسية مثل الدائرة والمستطيل والمثلث..الخ.

وبناء على البارمترات المرسلة عن طريق الرسائل لهذه الوحدة البرمجية يمكن التعرف عليها، ومن ثمّ يتم حساب المساحة المطلوبة.

التقنية الشيئية ومركزاتها:

تنظر التقنية الشيئية للعالم وكأنه مجموعة كائنات أو أشياء، حيث تقسم البرمجة الشيئية البرنامج إلى مجموعة كائنات وكل كائن يمثل جزء من حل المسألة.

ويتم التفاعل بين هذه الكائنات عن طريق ما يسمى الرسائل Messages.

وتتركز البرمجة الشيئية على الآتي:

1. تغليف البيانات والإجراءات في مكان واحد داخل الكائن والذي يسمى Encapsulation.
2. إضافة مفهوم التوريث Inheritence من كائن لآخر وهو أن الكائن يمكن له توريث خصائصه وعملياته(طرقه) لكائن آخر.
3. مفهوم تعدد الأشكال Polymorphism وهي تهدف لجعل الكائن العام يؤدي عدة وظائف.

مزايا وأهداف التقنية الشيئية:

من الأهداف المرجوة من استخدام التقنية الشيئية في التحليل والتصميم والبرمجة إنتاج برمجيات (منظومات) تتصف بالخصائص الآتية:

1. البرمجة بلغة طبيعية Natural language :

ونعني هنا استخدام مصطلحات طبيعية يفهمها كل من المبرمج والمستخدم ويبتعد قدر الإمكان عن مصطلحات البرمجة الاجرائية مثل المتغيرات Variables والمصفوفات Array. بتعبير آخر استخدام مصطلحات خاصة بالمسألة المراد حلها وليست مصطلحات خاصة بالحاسوب وهذا بالطبع يدعم إشراك الزبون في فريق إعداد المنظومة، وحتى في التصميم يجب ألا نغوص في التفاصيل البرمجية بل يمكن تمثيل المسألة بمستوى المتطلبات الوظيفية.

2. موثوقية (اعتمادية) البرمجيات المنجزة Reliability:

ونعني هنا إعداد برمجيات خالية من الأخطاء والإعطاب حتى يمكن الاستفادة منها بصورة جيدة، لهذا وجب عمل مجموعة اختبارات متنوعة لكل جزء أو مكوّن برمجي على حده.

3. إعادة الاستخدام Reuse

إذا تمت الاختبارات والمصادقة على أي جزء من المنظومة البرمجية فمن البديهي إعادة استخدام هذا الجزء بكل ثقة. ومفهوم إعادة الاستخدام يعتبر مفهوما قديما ومطبقا في التطبيقات الهندسية. فمثلا المهندس الكهربائي نراه يستخدم نفس المقاومة كلما أراد تصميم دائرة كهربائية مماثلة.

لهذا وبنفس المفهوم يمكن استخدام الكائنات في عدة برامج مستقبلية أخرى.

4. سهولة الصيانة Maintainable :

لاحظ أن من 60 ٪ إلى 80 ٪ من أعمال البرمجة تتم في الصيانة. لهذا نقول إن من الأهداف التي ترمي إليها التقنية الشيئية تقليل الصيانة إلى الحد الأدنى.

5. سهولة التمدد Extendable :

البرمجيات بصفة عامة غير ثابتة وتتغير باستمرار وتنمو عبر مرور الأيام حتى تتأقلم مع

البيئة وتبقى ذات فائدة وفاعلية . وكمثال على ذلك فقد يطلب المستخدم إضافة وظائف جديدة للمنظومة البرمجية.

6. تقليل مدة إعداد المنظومة البرمجية:

لقد كان إعداد البرمجيات في السابق يستغرق سنوات، أمّا الآن فتحسب مدة إعداد البرمجيات بالأشهر أو حتى الأسابيع، والذي ساعد على تقليص فترة البرمجة هو التقنية الشيئية التي وفرت برمجيات جاهزة ذات موثوقية وسهلة التمدد والاستخدام.

6.5 أدوات البرمجة Programming Tools

تعتبر لغات البرمجة من أهم الادوات المستخدمة في مرحلة التنفيذ، وننتقي عادة لغة مناسبة أو أكثر لكل تطبيق خاص، فمثلا قد نستخدم لغة فورتران للتطبيقات العلمية، ولغة كوبول للتطبيقات التجارية ولغة انترنت مثل ASP لتطبيق يستخدم الشبكة العنكبوتية.

6.5.1 لغات البرمجة:

توجد أكثر من 200 لغة برمجة، ولكن المشهور والمستخدم على نطاق واسع لا يتعدى العشرين لغة تقريبا.

ومن أهم هذه اللغات المستخدمة على سبيل المثال لا الحصر:

- لغة الفورتران FORTRAN تمّ استعمالها ولا زالت تستعمل للتطبيقات العلمية والهندسية.
- لغة C لغة مشهورة تستخدم في التعليم وبرمجة نظم التشغيل .
- لغة ++C لغة شيئية ذات صيت واسع وتستخدم في العديد من المؤسسات العلمية منها والتجارية، لأنها تقلل من وقت بناء المنظومات .
- لغة Visual Basic لغة مرئية ومسيرة بالحدث وهي تطوير للغة بيسك وتستخدم في برمجة العديد من التطبيقات المالية والإدارية والألعاب والانترنت نظرا لما تتميز به من واجهة مناسبة للمستخدم وبساطة في البرمجة.

- اللغات PhP JAVA, HTML and ASP، يستخدم بصورة خاصة فى تطبيقات الانترنت.
- نظم إدارة قواعد البياناتDBMS مثل لغة SQL (لغة الاستعلام المركبة) تعتبر لغة استفسار شبيهة بجمل اللغة الإنجليزية، وتستعمل عادة للتخزين والبحث والاسترجاع للسجلات والحقول فى قاعدة بيانات علائقية.
- لغة اوركل ORACLE لغة منافسة وتماثل لغة SQL من حيث أنها نظام إدارة لقواعد البيانات وهى تلقى رواجا فى إعداد البرمجيات .

6.5.2 أدوات مساعدة Utility Tools

تساعد هذه الأدوات في اعداد وتعديل وتصحيح وإدارة المنظومات البرمجية، ويمكن تصنيفها إلى

- المصححات Debuggers عبارة عن أدوات تستخدم لاكتشاف الأخطاء في البرامج لغرض تصحيحها.
- المحررات والمتصفحات Editors /Browsers تستخدم المحررات للتحديد والتركيز على أي جزء من الشفرة المراد تعديله.

وعادة ما تشمل المحررات ما يلي:

1. تلوين النص لغرض تحديده
2. القص واللصق بين النوافذ المتعددة
3. إلغاء التعديل وإعادته
4. اختيار العمود
5. الماكروات التي تكتب من قبل المستخدم لأداء مهام معينة

أما المتصفحات فتستخدم في تطبيقات الانترنت للبحث عن مواقع مختلفة في الشبكة العنكبوتية.

■ مكوّنات الشفرة Code Generators: تستخدم للتخلص من ضياع الوقت في كتابة التشفير. فهذه المكوّنات تمكّن المستخدم من تعريف الواجهة والتصميم، وبعض أدوات التصميم الاخرى فتتكون الشفرة تلقائيا.

وهذا يعني أن تصبح عملية البرمجة عملية ميكانيكية لا تحتاج إلى تركيز.

■ المكتبة The Library: تحتوي على العديد من الدوال والإجراءات التي تساعد المبرمج على إعداد تطبيقات برمجية ذات جودة وفي وقت قصير وباستخدام مكونات تمّ تخزينها مسبقا.

وفي النظم الشيئية OO systems يوجد ما يعرف بمكتبات الفصائل Class library التي تحتوي على فصائل يمكن أن تساعد في إعداد التطبيقات سريعا.

6.5.3 البرمجة بدون كتابة شفرة Programming Without Coding

هناك أنواع من اللغات تمكّن المبرمجين من إعداد براجهم دون أن يكتبوا سطرا واحدا من الشفرة.

وباستخدام هذه اللغات يحتاج المبرمج فقط إلى:

■ تعريف المسألة.

■ تصميم الحل.

■ استخدام الأدوات المناسبة لإيجاد الحل.

وبهذا بدلا من تعلم قواعد اللغة عليك فقط أن تتعلم استخدام طريقة عمل مجموعة من الأدوات، وأعتقد أنه بالرغم من ميزات هذه التقنية إلا أنها تضعف من الخلق والإبداع للعديد من المبرمجين، وشركة ميكروسوفت تتبع هذا النهج.

أكتب بعض الإرشادات لحرص كتابة شفرة جيدة

6.6 إرشادات في كتابة الشفرة الجيدة

الإرشادات التالية قد تساعد في الحصول على شفرة جيدة:

1. استعمل أقل ما يمكن من تراكيب التحكم if-then-else حتى يصبح البرنامج سهل القراءة والفهم والتعديل.

2. لا تستخدم جملة GO TO كل ما أمكن دائما، فهناك بدائل لها الآن.

3. لا تستعمل حلقات متداخلة Nested loop كثيرا، ويجب ألا يزيد عمق التداخل عن خمس تداخلات.

4. استعمل التنسيق Indentation والفراغات والأسطر الفارغة لغرض توضيح البرنامج.

5. يجب ألا يزيد البرنامج الفرعي عن 50 جملة.

6. من المستحسن أن تكون وحدة البرنامج ذات مدخل واحد ومخرج واحد.

6.7 برمجة المكونات Component Programming

إننا نعيش في عصر البرمجة الشيئية والنظم الشيئية . هذه الحقيقة غيرت من طرق إعداد وتصميم وتنفيذ التطبيقات . وبهذه التقنية يمكن أن تكون لدينا وحدات برمجية يمكن شراؤها وتعديلها حسب الحاجة.

إن القدرة على استعمال مكونات قابلة لإعادة الاستعمال لها أهمية خاصة في معمارية القطعة.

إنها تساعد في زيادة الابداع والإنتاجية بين المبرمجين.

وفي الحقبة الأولى من البرمجة حاول المبرمجون حفظ البرامج الفرعية بالمكتبة ولكن واجهتهم مشكلة في إعادة استعمال الشفرة.

أمّا الآن فاستطاعت التقنية الشيئية أن تجعل ذلك ممكنا عمليا.

والبرمجة الشيئية OOP تطوّرت إلى ما يعرف ببرمجة المكونات حيث يوجد لدينا مجموعة صغيرة من المبرمجين يعملون على إعداد المكونات.

وبناء المشروع من هذه المكونات يتطلب جهدا ووقتا أقل.

ونستطيع أن نقول إن البرمجة الشيئية قد ساعدت في حل جزء من أزمة البرمجيات ألا وهي مشكلة الإنتاجية وسرعة الإعداد.

أهم التطبيقات الصالحة للبرمجة وتصلح لأن تكون مشاريع برمجية:

1. المحاسبة.

2. الخطوط الجوية.
3. التعليم.
4. التدريب.
5. الصحة.
6. الاتصالات.
7. الجباية.
8. الرواتب.
9. نظام المخازن.
10. الصيدلة.
11. الرياضة.
12. الدعاية والإعلان.
13. الطباعة والنشر.
14. العقارات.
15. الوسائط المتعددة.
16. الكهرباء.
17. الهاتف.
18. الترفيه.
19. التصنيع.
20. الزراعة.

6.8 أسئلة للمراجعة

1. ما هو الفرق بين البرمجة الهيكلية والبرمجة الكائنية؟
2. عرّف:
   - Object
   - Class
   - Method
   - Attribute
   - Message
   - Encapsulation
   - Inheritance
   - Polymorphism
3. اكتب مثالا على :

Object-oriented Language: --

Structured language: --

Data Base Management System: --

Visual Programming Language: --

Internat Programming Language: --

4. لغة البرمجة ماهي إلا أداة واحدة للبرمجة. ما هي الأدوات الأخرى المتوفرة؟
5. اكتب بعض الإرشادات لغرض إعداد شفرة برمجة جيدة.

الفصل السابع

تأكيد الجودة واختبار البرمجيات

Software Quality Assurance & Testing



7.1 المقدمة

يعتبر تأكيد الجودة Quality Assurance من النشاطات المهمة والحيوية في أي مشروع تجارى ينتج منتوجات أو يقدم خدمات تستخدم من قبل الآخرين.

ومنذ الخمسينيات والستينيات كان تأكيد جودة المنتوجات البرمجية من مهمة المبرمج ولكن في بداية السبعينييات ظهرت هندسة البرمجيات كحل لمجموعة من المشكلات التى من أهمها ضعف الجودة، وأصبح من أهداف دراسة واستخدام هندسة البرمجيات إعداد منتوج برمجي فائق الجودة High Quality ومطابق للمواصفات والمعايير Standards المتفق عليها بين المؤسسات ذات العلاقة (مثل IEEE, ACM, ISO).

ومصطلح الجودة مصطلح معروف وقديم ومستخدم في إنتاج العتادHardware والمنتجات بصفة عامة . أمّا أهميته في إعداد البرمجيات فلم تتبين إلاّ في السنوات الأخيرة.

لقد كان الاهتمام في السابق وقبل السبعينيات مركزا على إنتاج منظومة دون أن تمر على اختبار كامل وشامل يؤكد خلوها تماما من الأخطاء، ونتج عن هذا التوجه مشاكل أدت إلى حصول ما يعرف بأزمة البرمجيات.

لهذا بدأ خبراء البرمجيات في اختيار توجه آخر يركّز على عملية الاختبار والاهتمام بالجودة. هذا التوجه يأخد في الاعتبار الاهتمام باختبار كل المنتجات التي يتم تجهيزها في كل مراحل إعداد المنظومة ابتداء من المراحل الأولى.

فظهر مصطلح تأكيد الجودة البرمجية Software Quality Assurance كحل للمشكلة.

ونظرا لأهمية الاختبار كأحد عناصر تأكيد الجودة نستعرض في هذا الفصل أنواع الاختبارات المختلفة خصوصا في هذه المرحلة الدقيقة والحساسة والتى تحدث بعد عملية البرمجة وقبل

تسليم المنظومة للمستخدم ليتم تشغيلها على أرض الواقع والاعتماد عليها في أداء الوظائف التي كانت تؤدى يدويا.

7.2 مفاهيم أساسية في الجودة وأهميتها

لكي نفهم مصطلح تأكيد الجودة للبرمجيات - QA Software Quality Assurance دعنا أولا نعرّج على مفهوم مشابه يسمى مراقبة الجودة Quality Control-QC والمستخدم في أغلب المصانع وتعتبر اليابان رائدة فيه خصوصا في صناعة المنتجات الالكترونية والسيارات وغيرها.

ومن أهمية هذا المصطلح تمّ استحداث أقسام في الشركات المصنعة تعرف باسم «مراقبة الجودة»، وتعتني بالمنتوج منذ تصنيعه وقبل وصوله للمستخدم، حيث تجري مجموعة اختبارات آلية ويدوية ومختلطة تتم عبر حلقة تسمى حلقة التغذية الاسترجاعية Feedback loop يتم فيها مقارنة المنتوج مع المواصفات والمعايير المحددة سلفا لهذا المنتوج.

إن مفهوم مراقبة الجودة يعطي الإدارة العليا المعلومات الضرورية (في شكل تقارير) بخصوص الجودة للحصول على الثقة في هذا المنتوج، والتأكد من أنه مطابق للأهداف والمواصفات والمعايير المطلوبة .

وبالمقابل إذا تمّ اكتشاف أيّ مشاكل فستكون الإدارة على علم مسبق، وفي وضع يؤهلها للتعاطي مع هذه المشاكل واستخدام الموارد والمستلزمات والإجراءات اللازمة والضرورية والكفيلة بحل هذه الاشكاليات الخاصة بالجودة في حينها.

وهناك ملاحظة مهمة وهي أن اكتشاف الأخطاء التي تظهر في المراحل الأولى تكون تكاليف إصلاحها أقل بكثير من تكاليف إصلاح الأخطاء التي تظهر في المراحل المتأخرة من المشروع.

لهذا وجب أن تبدأ عملية الاختبار منذ انطلاقة إعداد المتطلبات والمواصفات مع مواكبة مستمرة للجودة .

والهدف هو تقليل التكلفة لهذا المنتوج وجعل الزبون في وضع أفضل من الأمان والرضا.

مفهوم تأكيد جودة البرمجيات:

يمكن تعريف تأكيد جودة البرمجيات Software Quality Assurance- QA بأنه: «عملية التأكد من توافق المنتوج البرمجي مع المتطلبات الوظيفية والغير وظيفية وتطابق الأدوات والطرق والإجراءات والمنهجيات مع المعايير والمواصفات المعتمدة في إعداد البرمجيات».

والنقاط الآتية تؤكد هذا التعريف:

- المتطلبات Requirements الخاصة بالبرمجيات هي الأساس الذي تنطلق منه عملية قياس جودة المنتوج البرمجي.
- تحديد معايير ومواصفات مسبقة تساعد في تحديد معايير مطابقة للمواصفات المرغوبة أثناء عمليات تحديد المتطلبات والتصميم والبرمجة والاختبار وأن أي أنحراف وعدم تتبع واستخدام لهذه المعايير سيقودنا إلى نتائج لها مردود سيىء للجودة المنشودة.
- استخدام الأدوات والطرق والإجراءات في إعداد وصيانة المنتوج البرمجي ذو الجودة المطلوبة.

ويقوم عادة فريق متخصص بعملية تأكيد الجودة، وهو يختلف عن فريق إعداد البرمجيات، لكي تتم عملية اعتماد وقبول خصائص الجودة المطلوبة من وجهة نظر أكثر شمولية واستقلالية وحيادا، والأفضل أن يكون فريق تأكيد الجودة يمثل الزبون وينظر بمنظاره للتأكد من أن المنظومة تطابق معايير الجودة المنصوص عليها في وثيقة المتطلبات وخطة المشروع.

نشاطات تخص تأكيد الجودة:

هناك مجموعة مهام أو نشاطات تتم تأديتها من قبل فريق تأكيد الجودة تتلخص فيما يلي:

1. تجهيز خطة الجودة Quality plan حيث يتم تجهيزها في مرحلة التخطيط، وتتم مراجعتها من قبل ذوي الشأن، وبنود هذه الخطة تشمل:
   - تحديد البنود المراد تقييمها.
   - تحديد المراجعات المراد أداؤها.

- تحديد المعايير المراد التعاطي معها.
- تحديد الإجراءات المراد اتباعها والتي تخص تقرير الأخطاء.
- تحديد التقارير المراد إعدادها من قبل فريق إعداد الجودة.

2. مراجعة الأدوات والطرق والمفاهيم المستخدمة لإعداد المنظومة البرمجية والتأكد من توافقها مع معايير وسياسات المؤسسة صاحبة الشأن.
3. إفادة الإدارة العليا باستخدام التقارير المعدة سلفا لاتخاذ مايلزم.

ونورد فيما يلي أهم المراجعات المراد أداؤها وهي :

7.3 المراجعة والفحص والتصفح Reviewing, Inspection & Walkthrough

7.3.1 مراجعة البرمجيات Software Review،

تعتبر مراجعة البرمجيات Software Review مصفيا ضروريا ومهما لعملية إعداد البرمجيات حيث تتم مراجعة توثيق نهاية كل مرحلة: التحليل والتصميم والبرمجة.

والهدف هو كشف الأخطاء، وهي طريقة لضمان الجودة عن طريق فحص التوثيق للمنظومة تحت الإعداد لغرض إيجاد أي أخطاء فيه.

وفي المراجعة التي تتم بحضور الادارة يتم تجهيز تقرير يتم إعداده في نهاية كل مرحلة لاتخاذ قرار بخصوص المنظومة تحت الإعداد.

ونتيجة المراجعة يتم اتخاذ أحد القرارات التالية:

1. الاستمرار في المرحلة التي تليها.
2. التوقف عن المشروع بالكامل.
3. إجراء تغييرات على التوثيق ثمّ الاستمرار في المرحلة التى تليها.

والمراجعة بصفة عامة تعتبر طريقة تأكيد ومصادقة تحضرها الإدارة ويتم فيها:

- تحديد التحسينات الضرورية للمنتوج البرمجي الذي تمّ إعداده من قبل فريق إعداد المنظومة.
- تحديد وتأكيد الأجزاء التى لاحاجة ولا ضرورة لتحسينها.
- جعل العمل التقني أكثر ملاءمة ويتوافق مع ماهو متوقع.

7.3.2 فحص الشفرة Code Inspection

إلى جانب المراجعة على التوثيق في نهاية كل مرحلة يتم الفحص أو التفتيش Inspection على الشفرة Code.

فالفحص هو عملية لكشف الأخطاء في شفرة جزء برمجى وليس تصحيحها.

ومن أمثلة هذه الأخطاء: الأخطاء المنطقية في الشفرة.

ويتم فحص الشفرة في جلسة تعقد بحضور معد الجزء البرمجي وفريق الفحص (التفتيش) المكون من المنسق والشخص المراجع أوالمختبر وقارئ للجزء البرمجي. وتستغرق الجلسة الواحدة زهاء الساعتين.

ويتم توفير الآتي لهذه الجلسة :

- الشفرة المراد فحصها
- المعايير المستخدمة في إعداد المنظومة ليلم فريق الفحص بها وباستخدامها في عملية التفتيش.
- قائمة بالأخطاء المتوقعة Check list errors يتم تجهيزها بواسطة أخصائيين ذوي خبرة ويتم استخلاصها من منظومات سابقة.

ومن أمثلة بنود هذه القائمة مايلي:

1. هل تمّ إعطاء قيم ابتدائية للمتغيرات المستخدمة في الجزء البرمجي (الشفرة)؟
2. هل كل حلقة لها نهاية؟

3. هل عدد البارمترات المستخدمة في الإجراء أو الدالة صحيح ؟
4. هل التداخل بين جمل التحكم والحلقات صحيح؟
5. هل جمل التحكم مكتوبة بشكل صحيح وتوافق لغة البرمجة؟
6. هل جمل الإدخال والإخراج صحيحة؟
7. هل العمليات الحسابية في الجزء البرمجي تمت بطريقة صحيحة؟

هذا النوع من المراجعة قد يكلف الوقت والجهد، ولكنه يقلل من تكلفة الاختبارات بعد تنفيذ المنظومة.

وكما ذكرنا سابقا فان مهمة فريق الفحص هي تحديد واكتشاف الأخطاء وليس تصحيحها فالذي يقوم بتصحيح هذه الأخطاء هو نفس الشخص الذى أعد هذا الجزء البرمجي .

وتتم جلسة أخرى للتأكد من عدم اكتشاف أخطاء أخرى ناتجة عن تصحيح الأخطاء المكتشفة سابقا.

لذا نقول بأن عملية الفحص عملية تكرارية.

> **ملاحظة: رغم أن المراجعة والفحص لهما نفس الهدف، وهو إنتاج منتوج ذي جودة عالية، إلا أنهما يختلفان في طريقة العمل.**
>
> **تتم المراجعة عادة بصورة عامة، ويتم توجيه تقريرها إلى الإدارة لأخذ القرارات اللازمة حياله، ولكن الفحص يتم بشكل أكثر تفصيلا وتكرارا، ويتم توجيه التقرير الناتج عنه إلى معد الجزء البرمجي للتصحيح.**

7.3.3 التصفح Walkthrough

التصفح Walkthrough هو عبارة عن طريقة أخرى للمراجعة الغير رسمية وتهدف أصلا للحصول على الجودة في المنتوج البرمجي، شأنها شأن المراجعة والفحص، ولكنها تختلف عنهما بأنها عملية لمراجعة المنتوج البرمجي بواسطة الزملاء في فريق العمل Peers وذلك بهدف كشف الأخطاء (بدون تصحيحها) بطريقة غير منتظمة أي لاضرورة من معرفة مدير المشروع بها ولا ضرورة لتوثيقها بل تأتي عفوية بين أعضاء فريق المنظومة فيما بينهم.

فمثلا قد يعد أحد المحللين مخطط انسياب البياناتDFD وللتأكد من صحته قد يتصفحه ويراجعه محلل آخر من نفس الفريق لتحديد الأخطاء في هذا المخطط دون التعرض لطريقة تصحيحه.

والجدول التالي يعرض الفرق بين الفحص والتصفح :

التصفح Walkthrough	الفحص Inspection
يزيد من تواصل أعضاء فريق المشروع.	يستخدم قائمة أخطاء متوقعة
جلسات غير رسمية	تستخدم جلسات رسمية
جلسات غير منتظمة	أسلوب منتظم للبنود.
أداة تعليمية ممتازة لأعضاء الفريق الجدد.	يتم بواسطة عناصر مؤهلة.
يتم بواسطة زملاء فريق إعداد المنظومة	

7.4 التحقق والتصديق Verification & Validation

يستخدم التحقق Verification للتأكد من صحة توثيق المنتوج البرمجي في مراحل التحليل والتصميم والتشفير(على الورق بدون جهاز الحاسوب)، والتأكد من أن أي وثيقة في نهاية مرحلة ما مقتبسة من المرحلة السابقة لها.

فمثلا يمكن التأكد من أن شبه الشفرةPseudocode التي يتم إعدادها في مرحلة التصميم مقتبسة من المرحلة السابقة لها وهي مرحلة التحليل وبالتحديد مقتبسة من الإنجليزية الهيكلية Structured English أو حتى العربية الهيكلية Structured Arabic وهكذا. ويسمى التحقق بالكشف الساكن Static Checking او Static Testing الاختبار الساكن ويعنى الكشف عن الأخطاء على التوثيق (وثيقة المتطلبات أو وثيقة التصميم مثلا).

أمّا التصديق والاعتماد Validation فهو اختبار يتم عند تنفيذ المنظومةImplementation، ويتم من فريق عمل مستقل، حيث يجرى اختبار يسمى Validation test ويسمى أيضا

Dynamic Testing لأنه يجرى على جهاز الحاسوب (بدل التوثيق) للتأكد بأن المتطلبات الوظيفية والمدونة في وثيقة المتطلبات قد تمّ تنفيذها بالكامل وبطريقة صحيحة على جهاز الحاسوب.

والهدف كما أسلفنا سابقا الوصول الى برمجية ذات جودة عالية وتلبى متطلبات المستخدم بالكامل وبطريقة صحيحة ومرضية.

ونلخص الفرق بين الاعتماد والتحقق في الآتي :

■ **الاعتماد Validation**

هو عملية تقييم المنظومة في نهاية المشروع أي في نهاية مرحلة التنفيذ والتأكد من أن المنظومة تؤدي وظائفها طبقا للمتطلبات الواردة في وثيقة تحديد المتطلبات.

وتستخدم آلية اختبار الاعتماد على جهاز الحاسوب في عملية الاختبار وهو اعتماد ديناميكي.

■ **التحقق Verification**

هو عملية تحديد ما إذا كانت مواصفات مرحلة معينة (مثلا مرحلة التصميم) تحقق مواصفات مرحلة سابقة لها (مثلا مرحلة التحليل) أي مقتبسة منها وهو تحقق ساكن.

سؤال: رتب

التصفح، الفحص، التحقيق، المراجعة، الاعتماد حسب التسلسل الزمنى؟

الإجابة:

الترتيب حسب التسلسل الزمنى كالآتي :

1. التصفح Walkthrough
2. الفحص Inspection
3. التحقق Verification

4. الاعتماد والتصديق Validation
5. المراجعة Review

ملاحظة: يمكن مناقشة الموضوع مع أستاذ المادة لزيادة الفهم من قبل الطلاب واستخدام هذه الطرق في مشروع التخرج في شكل مجموعات من الطلبة.

قد ناقشنا في البنود السابقة بشيء من التفصيل الطرق التى توصلنا إلى الجودة.

أما ماهي خصائص الجودة المطلوبة في المنظومة فلم نوردها إلى حد الآن.

ولهذا نستعرضها في البند التالي تباعا:

7.5 خصائص الجودة Quality Attributes

اذكر 6 من خصائص الجودة

يجب أن تتصف المنظومة البرمجية الناجحة بالخصائص التالية:

1. الاعتمادية Reliability

وذلك بأن تقوم وتؤدي المنظومة ما يجب أن تقوم به من مهام ووظائف دون أي أخطاء أو عطل.

2. قابلية الصيانة Maintainability

وذلك بأن تكون المنظومة مصممة بطريقة تسهل عملية الصيانة .

3. التجزئة Modularity

وذلك بأن تكون المنظومة مجزأة بحيث يكون أي جزء برمجى شبه مستقل عن الأجزاء البرمجية الأخرى.

4. الأمن والحماية Security

وذلك بأن تكون للمنظومة جهاز حماية يمنع غير المخولين بالولوج إليها وتشمل حماية البيانات والبرامج.

5. التوسعية Expandability

وذلك بأن تكون المنظومة مفتوحة لإضافة مكونات جديدة مستقبلية أي يمكن التوسع في المنظومة من حيث البيانات والبرامج.

6. التكامل Completeness

وذلك بأن تكون كل الوظائف المطلوبة للمنظومة قد تم تنفيذها.

7. استقلالية عن العتاد Hardware Independence

وذلك بأن تكون البرمجيات مستقلة عن أي نوع من العتاد بحيث إذا تمّ تغيير في البرمجيات لايتطلب أي تغيير في العتاد.

8. البساطة Simplicity

وذلك بأن تكون المنظومة سهلة الفهم والتدرب عليها ولاتوجد أي صعوبة في استخدامها.

9. قابلية التتبع Traceability

أي قابلية تتبع الشفرة واقتباسها ورد أصلها إلى المتطلبات في مرحلة التحليل.

10. التوثيق Documentation

حيث يوجد بالمنظومة توثيق خاص بكل من معد المنظومة والمستخدم (مثلا دليل المستخدم).

11. الدقة Accuracy

وذلك بأن تكون المنظومة لها دقة في عملية إجراء الحسابات والتحكم.

12. قابلية التحويل Portability

يمكن تحويل أي جزء من المنظومة إلى بيئة جديدة مثل منظومة أخرى أو نظام تشغيل جديد أو حتى عتاد جديد.

13. الأداء Performance

أن تكون المنظومة سريعة الاستجابة (زمن الاستجابة في الاستفسارات سريع جدا) أو سريعة المعالجة .

14. سهولة الاستخدام Usability

أن تكون المنظومة سهلة الاستخدام وصدوقة للمستخدم User friendly وهذا يتطلب استخدام واجهة رسومية جذابة للمستخدم .Graphic User Interface

15. قابلية إعادة الاستخدام Reusability

أي أن أجزاء المنظومة يمكن إعادة استخدامها في تطبيقات (منظومات مستقبلية) أخرى Compenent Reuse .

16. المرونة Flexibility

أي أن المنظومة يمكن تطويرها وتعديلها بسهولة كلما لزم الأمر.

17. الكفاءة Efficiency

حيث يجب أن تقوم المنظومة بوظائفها بأقل مقدار من الموارد والمستلزمات.

18. قابلية الاختبار Testability

أي أن المنظومة يمكن اختبارها بمجهود غير مكلف للتأكد من تنفيذها لوظائفها كما هو مطلوب.

19. قابلية الفهم من قبل الآخرين Understandability

أي وضوح المنظومة من حيث التحليل والتصميم والشفرة والتوثيق.

20. الاقتصاد Economy

حيث يجب أن تكون المنظومة غير مكلفة في تشغيلها .

7.6 أنواع أخطاء البرمجيات

هناك ثلاثة أخطاء تحدث في المنظومات:

1. أخطاء في المتطلبات مثل وظائف المنظومة غير كاملة وغير دقيقة أو ناقصة.
2. أخطاء في التصميم مثل الإخفاق في ترجمة المتطلبات إلى مواصفات التصميم.
3. أخطاء في التنفيذ: مثل الأخطاء في إجراء العمليات الحسابية في منظومة المرتبات.

7.7 معايير البرمجيات Software Standards

7.7.1 أهداف المعايير

من المهم أن نضع بعض المعايير للبرمجيات الجيدة للأهداف الآتية:

- للتفاهم بين متخصصي الحاسوب والزبون وللوصول لعمل مثالي، وفي نفس الوقت لإعطاء شعوربالقناعة والرضا لدى الزبون.
- لتجنب تكرار أخطاء الماضي.
- لاستمرارية العمل في المنظومة حتى في غياب بعض أفراد فريق إعداد المنظومة.
- للتأكد من أن المهندسين والمبرمجين في المؤسسة الواحدة يستعملون نفس الأدوات والتقنيات.
- لتقليل زمن التدريب عند البدء في مشروع جديد.
- للحصول على منتوج ذي جودة عالية.

7.7.2 أنواع المعايير Types of Standards

1. معايير خاصة بعملية إعداد المنظومة Process standards

وهي معايير تتعلق بالطرق والأدوات والفنيات المستخدمة في إعداد البرمجيات.

أمثلة :

- تحديد المتطلبات يجب أن يكون كاملا قبل التنفيذ.
- عدم استخدام النموذج التجريبي Prototype لأنه من المحتمل أن يسيء الزبون فهم النموذج التجريبي باعتباره المنظومة النهائية الكاملة.

2. معايير خاصة بالمنتوج البرمجي Product Standards

تتعامل هذه المعايير مع شكل المنتوج البرمجي وليس مع كيفية إعداده أو الأدوات التي استخدمت.

أمثلة :

- معايير في تسمية الوثائق المستخدمة في المنظومة.
- معايير في شكل وهيئة شفرة البرنامج.
- معايير في شكل وبنود خطة المشروع.

7.7.3 معايير البرمجة Programming Standards

اذكر ثلاث أمثلة لمعايير البرمجة

أمثلة :

- يجب عدم استعمال جملة GO TO كلما أمكن ذلك .
- يجب ألا يزيد عمق تداخل الحلقات عن 5 مستويات.
- يجب ألا تزيد شفرة البرامج الجزئية (الفرعية) عن 40 سطرا .
- استخدام تنسيق مناسب في كتابة الشفرة (مثل كتابة تداخل الحلقات أو جمل التحكم بطريقة واضحة وغير قابلة للالتباس).
- استعمال أسماء متغيرات ذات مدلول واضح.
- يجب أن تنفذ البرامج باستعمال مترجم مناسب .

- استعمل نظام تشغيل متداول.
- استعمل بروتوكولات متداولة ونظام ملفات شبكة ذات معايير قياسية .

7.8 الاختبار Testing

هو مجموعة من الأنشطة activities التي إذا تم تخطيطها وتنفيذها بصورة صحيحة أثناء إعداد البرمجيات فإنها ستؤدي إلى منظومة موثوق منها reliable system خالية من الأخطاء وتعطي نتائج سليمة.

وفي هذا الباب سنوضح عددا من طرق وأدوات واستراتيجيات الاختبارالتي يتم استخدامها عند إعداد منظومة برمجية ذات مواصفات مناسبة.

أنواع الاختبار Types of Testing

يتم اجراء الاختبارات من قبل عدة أنواع من الأفراد سواء من فريق إعداد المنظومة أو خارج هذا الفريق، وبعض هذه الاختبارات يجب أن يتم عن طريق فريق متخصص من جهة مستقلة ومحايدة والهدف هو الحصول على منظومة سليمة الأداء وبدون أخطاء إلى أبعد حد.

والجدير بالذكربأن الاختبارات على المنظومة وخصوصا الكبيرة والمعقدة منها تستغرق حوالى 40 ٪ من الزمن الكلي لإعداد المنظومة وهذا يجعل عملية الاختبار ذات أهمية بالغة.

ويجب أن يعمل كل من معد المنظومة البرمجية ومجموعة الاختبار المستقلة جنبا إلى جنب، للتأكد من خلو المنظومة من الأخطاء وتنفيذ الاختبارات الكافية لتحقيق هذا الهدف.

وكما ذكرنا سابقا فإن المستخدم يجب أن يلعب دورا مهما في عملية الاختبار وبالتحديد في اختبار القبول (اختبار المستخدم).

ومن هذه الاختبارات مايلي:

1. اختبار الجزء البرمجى Module Test

اختبار الجزء البرمجى Module Test (يسمى أيضا باختبار الوحدة) Unit Test يركّز على

اختبار أصغر وحدة في النظام Module اي الجزء البرمجي، وعادة مايتم هذا الاختبار أثناء عملية البرمجة أي في المرحلة السابقة لمرحلة الاختبار من قبل المبرمج المختص بإعداد وتنفيذ هذا الجزء البرمجي.

وعلى هذا يمكن القول بأن كل مبرمج يجب أن يقوم بتنفيذ اختبار الجزء الخاص والمكلف به بمفرده ويجب تسليمه لمدير البرمجة بدون أخطاء وأن يكون صالحا للاستعمال وجاهزا للاختبار التالي ألا وهو اختبار التكامل.

ما الفرق بين واختبار القبول

2. اختبار التكامل Integration Test

تجمع في اختبار التكامل Integration Test الأجزاء البرمجية التي تم إعدادها واختبارها من قبل مبرمجين فرادى في مرحلة البرمجة السابقة لتكوّن رزمة برمجية كاملة من قبل مدير البرمجة وبعدها يتم تركيب واختبار هذه الوحدات كوحدة واحدة حيث يتم اكتشاف الأخطاء وتصحيحها بسهولة، كما يتم اختبار البينياتinterfaces بصورة كاملة.

فمثلا إذا افترضنا أن المنظومة مقسمة إلى مجموعة أجزاء برمجية: جزء برمجي لإضافة صنف وجزء آخر للاستفسار وجزء ثالث لتعديل بيانات الصنف فهنا يقوم كبير المبرجين بإجراء عملية الاختبار على النحو التالي :

إضافة صنف بإدخال بيانات ذلك الصنف عن طريق لوحة المفاتيح لتكوين سجل خاص بهذا الصنف ثم الاستفسار عنه ليراه على الشاشة ثم يقوم بتعديل أحد الحقول وليكن سعر الصنف ثم يقوم مرة أخرى بالاستفسار ليرى هل تم تعديل هذا السعر حقا لنفس الصنف أم لا، وفي حالة الاجابة بنعم تكون هذه الأجزاء البرمجية سليمة ويعتبر هذا الاختبارناجحا.

3. اختبار المصادقة Validation Test

يوصف هذا النوع من الاختبار بالحركي لأنه يتم عبر جهاز الحاسوب وليس عبر الوثائق، ويتم أداؤه من قبل فرد مستقل أو جهة مستقلة (مثلا مكتب اختبار استشاري) ويركز على إثبات وتعزيز صحة المنظومة والتثبت من أن وظائف المنظومة البرمجية التى تم

إعدادها ويجرى الاختبار عليها تلبي المتطلبات الوظيفية(الوظائف) كما جاءت في مستند مواصفات المتطلبات requirement specification document .

4. اختبار القبول Acceptance Test

يسمى هذا الاختبار بعدة مسمّيات، منها على سبيل المثال اختبار المستخدم User test حيث يقوم هذا المستخدم بإجراء هذا الاختبار للتأكد من صلاحية المنظومة للعمل ليتم اتخاذ القرار بخصوص قبول المنظومة من عدمه.

ويقوم المستخدم عادة بتجريب المنظومة أولا في مكان معد المنظومة ويسمى هذا الاختبار باختبار ألفا alpha test ثم بعد ذلك يتم تركيب المنظومة في مكان او موقع الزبون حيث يتم إجراء الاختبار عليها ويسمى هذا باختبار بيتا beta test، وهو يتعلق بالاختبارات التالية:

- اختبار الوظائف Functional Test

اختبارات تتم من قبل الزبون أو بمساعدة طرف خارجي يختاره ويتم فيها التأكد من أن الوظيفة المراد اختبارها ولتكن مثلا "إضافة سجل صنف" في منظومة مخازن تقوم بأداء هذه الوظيفة بصورة صحيحة وبدون أخطاء.

- اختبار الكفاءة Performance Test

يسمى هذا الاختبار ايضا باختبار زمن الاستجابة response time test والغرض منه التأكد من سرعة وظيفة أو معاملة معينة في المنظومة أى سرعة زمن الاستجابة ولتكن مثلا «الاستفسار عن بيانات سجل موظف» في منظومة الموظفين وعرض هذه البيانات على شاشة جهاز الحاسوب بعد إدخال رقم الموظف، ثم تحديد وحساب زمن الاستجابة (أي سرعة عرض البيانات على الشاشة) بحيث لايزيد هذا الزمن عن الزمن المتفق عليه سلفا في وثيقة مواصفات المتطلبات تحت بند متطلبات الأداء، ويتم في هذا الاختبار أيضا التأكد من أن المنظومة لها القدرة التخزينية على استيعاب الكم الهائل من السجلات الخاصة بقاعدة البيانات.

5. اختبار الإجهاد Stress Test

يتم تجهيز مجموعة حالات اختبار غير طبيعية بغية حصول الطمأنينة من أداء المنظومة في الظروف الغير الاعتيادية.

ويسمى في هذه الحالة باختبار الإجهاد لتجربة أداء المنظومة بهدف التأكد من عمل المنظومة تحت ظروف وضغط متزايد في حالات استثنائية، ومحاولة ملاحظة تعامل وتعاطي المنظومة مع هذه الظروف الصعبة ومن ثمّ تعديل المنظومة لمجابهة هذه الظروف والتعامل معها مع المحافظة على الأداء الجيد مثل :

- تشغيل عدد أكثر من الطرفيات (شاشات أو أجهزة شخصية مربوطة في شبكة) من العدد المسموح به في الشبكة.
- قطع إحدى حلقات الاتصال مثل قطع التيار الكهربائي فجأة أو فصل أحد خطوط الشبكة وملاحظة سلوك المنظومة مع هذه الحالات المتطرفة.

6. اختبار الصندوق الأسود Black-Box Testing

يتم هذا الاختبار على أي جزء برمجي بتنفيذه على الحاسوب، وهو اختبار وظيفي يتعلق بمعرفة نتائج الاختبار عند إدخال بيانات معينة ومحددة.

أي أننا في هذا الاختبار ننظر إلى الجزء البرمجي وكأنه صندوق مصمت ونهتم فقط بالمدخلات والمخرجات دون الاهتمام بتفاصيل هذا الجزء البرمجي.

فمثلا لاختبار جزء برمجي يقوم بترتيب مجموعة من الأسماء، فإننا في هذه الحالة سنركز على أن النتائج تكون مجموعة من الأسماء المرتبة دون الاهتمام بالطريقة(أو الخوارزمية) التي تم بها الترتيب.

7. اختبار الصندوق الأبيض White-Box Testing

يتم في اختبار الصندوق الأبيض White-Box Testing ويسمى أيضا باختبار الصندوق الزجاجي Glass Box تشغيل المنظومة بيانات مختلفة ومتنوعة وذلك للتأكد من أن كل مسارات الخوارزمية للجزء البرمجي الواحد قد تمّ الوصول إليها والتأكد من سلامة أدائها.

8. اختبار ألفا Alpha Testing

يتم اختبار ألفا Alpha Testing من قبل المستخدم في مكان معد المنظومة developer site والهدف هو اكتشاف الأخطاء التي لا يكتشفها إلا المستخدم النهائي end-user ويبتدىء الاختبار بتجريب المنظومة بواسطة المستخدم على جهاز الحاسوب حيث يراقب معد المنظومة الاختبار في بيئة محكمة ويسجل الأخطاء التى يتم اكتشافها.

ثم يتم إصدار نسخة أخرى متضمنة تعديلات طبقا لهذه الملاحظات وتعاد الكرة مرة أخرى أي الاختبار من قبل المستخدم للمنظومة مع إبدا الملاحظة من قبل معد المنظومة إلى أن يتم الوصول إلى منظومة تلبي احتياجات المستخدم.

9. اختبار بيتا Beta Testing

بعد الانتهاء من اختبار ألفا في مقر معد المنظومة developer site، يبدأ اختبار بيتا beta test في مقر الزبون customer site حيث يسلّم معد المنظومة نسخة جديدة new version من المنظومة للمستخدم ليختبرها بمفرده وفي موقعه ويسجل ملاحظاته عن أي أخطاء صادفته ثم يسلمها ثانية إلى معد المنظومة للتصحيح وإصدار نسخة معدّلة وإرجاعها ثانية للمستخدم وهكذا.

وباستخدام هذا النوع من الاختبار يمكن إبراز وإظهار مشاكل مختفية على السطح لم يتم التنبه إليها أو ملاحظتها في الاختبارات السابقة لأن المستخدم سيستعمل طريقته الخاصة في تشغيل المنظومة وقد يخطىء عفويا في عملية الإدخال مثل ادخال حروف بدل أرقام أو الضغط على زر من أزرار لوحة المفاتيح بطريقة خاطئة.

أي أن هذا الاختبار ينقح الشفرة ويوجه المنظومة إلى وجهة صحيحة وناجحة وتحت بيئة واقعية.

ويجب على معد المنظومة أن يأتي عدة مرات للمستخدم لغرض تجميع الملاحظات والأخطاء ليتم التعامل معها وفي فترات زمنية منتظمة (مثلا مرة كل أسبوع) وذلك لغرض إصلاحها وإعداد نسخة جديدة من المنظومة وتسليمها للمستخدم مرة اخرى وتستمر

هذه العملية حتى يتم إصلاح كل الأخطاء وتبدأ المنظومة في وضع أفضل لقبولها ومن ث[illegible] الموافقة عليها والبدء في تشغيلها.

لذلك سمي هذا الاختبار باختبار القبول acceptance test.

10. اختبار التحقق Verification Testing

كما أسلفنا سابقا، فان هذا النوع من الاختبار لا يتطلب تنفيذه استخدام جهاز الحاسوب لأنه يتعامل مع الوثائق المعدة في نهاية كل مرحلة حيث يتم في هذا الاختبار التأكد من أن أي وثيقة مواصفات في أي مرحلة من مراحل المشروع مقتبسة ومرتبطة بالوثيقة في المرحلة السابقة لها.

فمثلا يجب أن يكون المخطط الهيكلي structure chart الذى يتم إعداده في مرحلة التصميم مقتبسا ومتحولا من مخطط انسياب البيانات Data Flow Diagram في مرحلة التحليل.

وهذا النوع (بعكس اختبار المصادقة validation الذي يتم تنفيذه على الحاسوب) يوصف بأنه ستاتيكي (ساكن) بينما يوصف اختبار المصادقة بأنه ديناميكي (حركي).

ويتم في هذا الاختبار فحص وثيقة مواصفات المتطلبات، ووثيقة مواصفات التصميم ووثيقة شفرة المصدر source code التي يتم تجهيزها في نهاية مراحل التحليل والتصميم والبرمجة، وتحديد واكتشاف الأخطاء الواردة بهذه الوثائق ليتم ردها مرة اخرى لمعديها لتصحيحها.

وبهذه الطريقة يمكن اكتشاف الأخطاء بسرعة أكبر وبتكلفه أقل لأن 50 ٪ من الأخطاء يمكن أن تكتشف قبل التنفيذ.

لاحظ مرة أخرى إن اختبارات المصادقة والتحقق تتم عادة بواسطة مكاتب متخصصة تسمى IV&V وتعنى verification &independent validation أي المصادقة والتحقق المستقل ومعناها أن الاختبارات تتم من قبل فريق مختلف عن فريق إعداد المنظومة.

11. اختبار الحماية Security Testing

يستخدم اختبار الحماية Security Testing للتأكد من أن آلية الحمايه للمنظومة تمنع دخول أي مستخدم غير مخوّل او مسموح بدخوله إلى المنظومة (ولوج غير قانوني) لأي سبب أو غرض.

12. اختبار الاسترداد Recovery Testing

يبرهن اختبار الاسترداد Recovery Testing على أن المنظومة يمكن ان تسترد عافيتها في حالة حدوث مشكلة أو حدث طارىء وتستمر في العمل لوقت محدد مسبقا.

وهذا الاختبار يجبر المنظومة في الوقوع في مشكلة قد تسبب في توقفها بطرق مختلفة، ويتأكد فني الاختبار من أن المنظومة تستطيع ان ترجع لعملها بدون تلف البيانات المخزنة.

وكمثال على اختبار الاسترداد قطع الكهرباء علي جهاز الحاسوب ثم التأكد من استطاعة المنظومة على أن تحقق عملية الاسترداد خلال وقت محدد من الزمن.

ومن أمثلة هذا الاختبار:

1. إعادة التشغيل المنظومة Reinitialization.
2. التخزين التلقائي في حالة حصول عارض.

7.9. أسئلة مراجعة

1. ماذا يعني مصطلح QA ومصطلح QC ؟
2. اذكر ستة من خصائص الجودة للمنتوج البرمجي.
3. املأ الفراغات الآتية:

الاعتمادية تعني أن يقوم النظام بعمله المطلوب دون أي أخطاء أو أعطال أما الأداء Performance فهو يشمل:

- سرعة الاستجابة
- سرعة المعالجة

4. قابلية التحويل portability تعني تحويل أي جزء من المنظومة من بيئة إلى أخرى.
5. عرف المصطلحات التالية
   - Inspection
   - Walkthrough
   - Review
6. مالفرق بين التصفح والتفتيش؟
7. ضع حرف T للجمل الصحيحة وحرف F للجمل الخاطئة:
   - يتم توجيه تقرير QA إلى معد المنظومة ( ).
   - تتم عملية V&V من قبل مجموعة مستقلة ( ).
   - العمليتان QA وV&V لهما نفس الهدف، وهو التأكد من أن خصائص المنتوج والعملية مرضية ( ).
8. ما معنى المصادقة وما معنى التحقق؟
9. اكتب 3 أمثلة لمعايير البرمجة.
10. اكتب بإيجاز عن:
    - Unit testing
    - Integration testing
    - User testing
11. قارن:
    - بين اختبار الصندوق الأسود والصندوق الأبيض
    - اختبار ألفا وبين اختبار بيتا.

12. لماذا يوصف اختبار التحقق بأنه استاتيكي؟

13. اكتب مثالا على الاختبار الديناميكي وما هو الاسم المرادف ؟

14. اشرح معنى المصطلحات الآتية :

- Security Test اختبار الحماية
- Recovery Test // الاسترداد

15. ما الفرق بين اختبار القبول واختبار التكامل؟

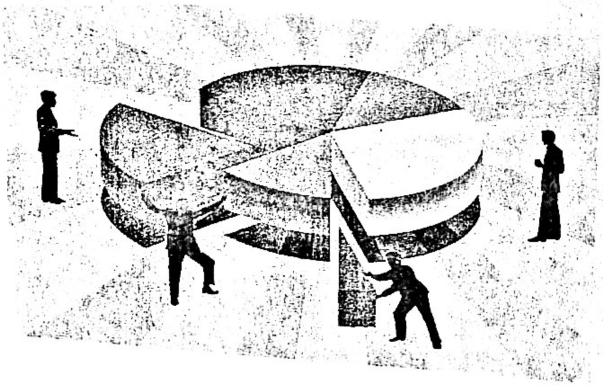
16. بيّن أي من الجمل التالية صحيحة وأيها خاطئة مع ذكر السبب :

- اختبار الصندوق الأسود يكافئ الاختبار الوظيفي.
- اختبار الصندوق الزجاجي يكافئ اختبار الصندوق الأبيض.
- اختبار الحماية يعني عدم السماح بالدخول للمنظومة.
- اختبار ألفا واختبار بيتا يجريان بواسطة المستخدم في موقعه.

الفصل الثامن

نماذج إعداد المنظومات

Software Development Models



8.1 مقدمة

يعتبر هذا الفصل من الفصول المهمّة في هندسة البرمجيات لأنه يتناول النماذج المختلفة المستخدمة في إعداد المنظومات، والتي ساهمت مساهمة جدية للتعامل مع هذا الكم الهائل من المتطلبات الجديدة والمتنوعة والمعقدة في بعض الأحيان، والتعامل مع أزمة البرمجيات التي أرّقت العلماء وجعلتهم في سباق مع الزمن لإنهائها.

ففي الخمسينيات وإلى نهاية السبعينيات كان إعداد المنظومات يتم بطريقة بدائية، حيث لا يوجد توثيق يذكر ولا أدوات جيدة في مراحل التحليل والتصميم والبرمجة ولا يوجد شكل مشروع بالمفهوم السائد الآن.

لقد سمّي النموذج البدائي باسم برمج وصحح Build & Fix أي اكتب البرنامج وادخله لجهاز الحاسوب ثم صحح الأخطاء وجربه إلى أن تصل إلى الحل النهائي الصحيح وكانت المتطلبات في رأس المبرمج ولا يوجد فريق عمل بالمعنى الصحيح للكلمة.

لهذا كانت مرحلة صعبة جعلت المتخصصين يستعينون بالعلوم الأخرى.

ومنذ بداية السبعينيات بدأ إعداد المنظومات يتم بطريقة صحيحة وفي شكل مشروع أسوة بالمشاريع الهندسية الأخرى، وهذا المشروع يتكون من مجموعة مراحل عرفت باسم دورة حياة إعداد المنظومة Software Development Life Cycle.

واستخدمت بداية من ذلك التاريخ مفاهيم وطرق ومنهجيات وأدوات وتقنيات جديدة لتواكب هذا التوجه الجديد والذي تبنى المنهاج الهندسي.

وسنتعرض في البنود التالية للعديد من النماذج المختلفة لإعداد المنظومات التي تم تطويرها خلال الفترة من السبعينات إلى أواخر التسعينيات.

وكل نموذج له محاولة متقدمة للمساعدة في الوصول إلى مشروع برمجي ناجح .

و في هذا الباب سنقدم بعض هذه النماذج التي تستخدم أنواعا مختلفة من الأدوات tools والطرق Methods للحصول على أفضل الحلول لمشاكل إعداد البرمجيات.

8.2 النموذج التدفقي Waterfall model

يسمي النموذج الشلال أو التدفقي Waterfall model (وهو أول نموذج تم استخدامه بداية من السبعينيات) أيضا بالنموذج التسلسلي Sequential model نظرا لتتابع المراحل وكان طوق النجاة في نهاية الستينيات، وله الفضل في حل الكثير من المشاكل التي تمت مواجهتها عند إعداد المنظومات، وساهم إلى حد ما في حل مشكلة أزمة البرمجيات Software Crisis حيث كانت المنظومات يتم إعدادها بدون توثيق وتستخدم أدوات بدائية في التحليل والتصميم.

وكان المخطط الانسيابي Flowchart الأداة الوحيدة المستخدمة، وبذلك كانت الجودة متدنية والأخطاء كثيرة، والزبون في أغلب الاحيان غير راض عن المنظومة النهائية.

ويشترط استخدام هذا النموذج أن تكون المتطلبات واضحة وكاملة.

وقد تمّ استخدام هذا النموذج كثيرا في السابق في إعداد المنظومات، وهو مقتبس من الطريقة الهندسية في التصنيع والإنتاج والبناء من حيث عملية المراحل واستخدام مفهوم المشروع.

أما المراحل التي تخص النموذج التدفقي فهي:

1. مرحلة التخطيط Planning phase
2. مرحلة التحليل Analysis phase
3. مرحلة التصميم Design phase
4. مرحلة التنفيذ Implentation phase
5. مرحلة الاختبار Testing phase
6. مرحلة الصيانةMaintenance phase

ونستعرض هنا ملخصا لأهم النشاطات التي يتم أداؤها في كل مرحلة من مراحل النموذج التدفقي:

1. مرحلة التخطيط Planning phase:

يبدأ التخطيط من اللحظة التي يشعر فيها الزبون بالحاجة Need والرغبة في الحصول على حل لمشكلة قائمة، وبالتالي يقوم بإعداد طلب (على شكل رسالة أو وثيقة) إلى الإدارة المسؤولة يلتمس فيه التغيير إلى الأفضل.

ويسمى هذا بالالتماس User request وفيه يتعرض الشخص أو القسم المختص صاحب المشكلة إلى مايلي:

- ملخص للمنظومة الحالية Probem statement.
- الأهداف المرجوة من التغيير المطلوب Objectives.
- نطاق العمل المراد تنفيذهScope .

ويتضمن هذا الطلب الرغبة في الحصول على أحد الاختيارات الآتية :

- إعداد منظومة جديدة.
- تغيير طفيف أو تصحيح أخطاء في النظام الحالي.
- إضافة بعض الوظائف للمنظومة الحالية.

وفي هذه المرحلة يتم أيضا عمل دراسة جدوى feasibility study للتأكد من جدوى هذا الالتماس، وبالتالي تخصيص موارد مالية لهذا المشروع .

وفي نهاية هذه المرحلة يتم إعداد خطة للمشروع Project plan تتضمن الأهداف المرجوة والجدول الزمني والمستلزمات المطلوبة.

2. مرحلة التحليل :Analysis phase

إن الهدف الأساسي من مرحلة التحليل هو فهم متطلبات المستخدم User Requiement

وتنظيم وتمثيل Modeling هذه المتطلبات في شكل مخططات وجداول لتتم الموافقة عليها مع الزبون واعتبارها اساسا للمرحلة اللاحقة ألا وهي مرحلة التصميم.

وفي مرحلة التحليل يتم أيضا جمع الحقائق حول النظام الحالي وفهم المشاكل التي تعترض سير العمل، وتستخدم عدة طرق للوصول لهذا الفهم مثل المقابلة الشخصية والاستبيان والملاحظة والعرض التجريبي وغيره ومن ثمّ تحديد المتطلبات الوظيفية Functional requirement وغير الوظيفية non-functional requirements للمنظومة المقترح تنفيذها.

ويمكن إعداد المنظومة باستخدام أحد المنهجيات الآتية:

- المنهجية الهيكلية Structured Methodology: تقسم هذه المنهجية المنظومة إلى أجزاء وظيفية في شكل هرمي وتستخدم أدوات هيكلية في التحليل والتصميم والبرمجة مثل مخطط انسياب البيانات DFD والمخطط الهيكلي Structure chart
- المنهجية الشيئية Object Oriented Methodology: تقسم هذه المنهجية النظام إلى كائنات لها وظائف وخصائص شبيهة بالكائنات التي نتعامل معها في حياتنا اليومية. وتستخدم أدوات شيئية مثل مخطط استخدام الحالة UCD ومخطط الفصيلة class diagram .
- منهجية فصائل المكتبة: Class library حيث يستطيع من خلالها معد المنظومة تجهيز منظومة تجريبية أو إعداد منظومة متكاملة في خلال فترة وجيزة (ساعات أو أيام) .

وتحتاج بالطبع إلى التدرب عليها، وتحتوي على الأدوات الضرورية لتحليل وتصميم وتنفيذ المنظومات مشتملة أيضا على كيفية تصميم واجهة المستخدم.

ومن أمثلة هذه الفصائل مايسمى الفصائل الأساسية لميكروسوفت Microsoft Foundation Classes MFC: وهي عبارة عن مكتبة فصائل وتسمى أيضا Application Frameworks .

وتتلخص أهم نشاطات التحليل فيما يلي:

- اكتشاف المجهول من قبل محلل النظم . لهذا وجب عليه في البداية الاستماع للزبون والمستخدم وملاحظة مجريات الأمور في النظام الحالي.

- الاستقصاء والتنقيح والفهم لمشاكل النظام الحالي وتحديد المتطلبات للنظام الجديد.
- تمثيل مافهمه بالمخططات وشاشات الحوار والتثبت من البيانات المأخوذة بعناية.
- بعد فهم المسألة تأتي كتابة وثيقة مواصفات المتطلبات لتصبح على شكل عقد أو اتفاقية بين المحلل والزبون للأشياء المطلوب تنفيذها في المنظومة المقترحة.

3. مرحلة التصميم Design phase

تأتي هذه المرحلة بعد تحديد المتطلبات من قبل الزبون، وإعداد مواصفات دقيقة لهذه المتطلبات.

حيث يتم في هذه المرحلة(مرحلة التصميم) تمثيل الحل في شكل خرائط وجداول ومخططات تفصيلية نهائية blueprint للمنظومة المقترحة التي تلبي المتطلبات المدونة في وثيقة مواصفات المتطلبات في المرحلة السابقة لها.

وتستخدم هذه المرحلة أدوات تماثل وتشبه الخرائط الانشائية والمعمارية التي تسبق عملية البناء. وفي هذه المرحلة يتم:

- تحديد المدخلات Input
- تحديد المخرجات Output
- تحديد البينيات Interface مع المنظومة التي تشتمل على:
  - بينية البرمجيات.
  - بينية العتاد.
  - بينية واجهة المستخدم.
- كتابة للخوارزميات Algorithms.
- تصميم البيانات الخاصة بالمنظومة Data design.

وفي نهاية مرحلة التصميم يتم إعداد وثيقة تسمى وثيقة مواصفات التصميم Design

specification document لتتم مراجعتها من قبل الإدارة واعتمادها لتصبح الأساس للمرحلة اللاحقة ألا وهي مرحلة التنفيذ أو مرحلة البرمجة.

4. مرحلة التنفيذ Implementation phase

يتم في هذه المرحلة برمجة المنظومة باستخدام إحدى لغات قاعدة البيانات واللغات الأخرى المناسبة لنوع التطبيق تحت الإعداد.

فيتم توزيع الوحدات البرمجية على مبرمجي المنظومة من قبل مدير المشروع، بحيث أن كل مبرمج يترجم الخوارزمية الخاصة بالجزء المنوط ببرمجته ويتمّ ادخالها للحاسوب ومن ثمّ اكتشاف وتصحيح الأخطاء Debugging، ثم اختبار الوحدة البرمجية Module test وهي اختبار يتم فيه التأكد من أن هذا الجزء يؤدي وظيفته بطريقة صحيحة.

5. مرحلة الاختبار Testing phase

تعتبرهذه المرحلة من أطول المراحل وأهمها (قد تستغرق من 30 ٪ إلى 40 ٪ من الزمن الكلي لإعداد المنظومة)، لأنها تتعلق بجودة المنظومة والتأكد من استيفاء المنظومة لجميع المتطلبات وبطريقة صحيحة ودقيقة وكاملة والتي ينتج عنها قبول أو عدم قبول المنظومة من قبل الزبون.

وتشتمل هذه المرحلة على مجموعة اختبارات عديدة قد تربو على عشرة أختبارات والتي من أهمها:

- اختبار التكامل Integration test.
- اختبار النظام System test.
- اختبار المستخدم او اختبار القبول User test or Acceptance test.

6. مرحلة الصيانة Maintenance phase

تعتبر مرحلة الصيانة Maintenance من المراحل المهمة في حياة المنظومة التي يجب أن نوليها الاهتمام الكامل إذا أردنا أن تعيش فترة طويلة من الزمن (تقدر بحوالي 10 إلى 15 سنة).

وتحدث الصيانة في المنظومة البرمجية نتيجة لعدة ظروف مختلفة نوجزها فيما يلي:

1- 65 ٪ من عمل الصيانة يتم نتيجة إضافة وظائف جديدة أو تعديلها

2- 18 ٪ من عمل الصيانة يتم نتيجة تكيف المنظومة مع تغير في البيئة التي تتعلق بالمنظومة.

3- 17 ٪ من عمل الصيانة يتم نتيجة تصحيح أخطاء أثناء تشغيل المنظومة

وتسمى هذه المرحلة أيضا بمرحلة الاستعمال والدعم للمنظومة Operation and support. وتبدأ منذ لحظة تسليم المنظومة للزبون والبدأ في تشغيلها على أرض الواقع.

ومن أهم النشاطات التي يمكن أن تحدث أثناء هذه المرحلة مايلي:

1- تصحيح الأخطاء التي قد تحدث أثناء تشغيل المنظومة

2- تعديل المنظومة حتى تتكيف مع التغيرات التي تحصل بتغير القوانين او تغير التقنية المستخدمة في العتاد أو البرمجيات

3- تغيير المنظومة نتيجة طلب الزبون بإضافة وظائف أو مزايا للمنظومة مثل طباعة الصك المودع في منظومة المصارف أو طباعة ايصال الإيداع.

هناك أربع أنواع من الصيانة البرمجية تحدث أثناء عمل المنظومة في البيئة العملية وهي:

- صيانة لتصحيح الأخطاء Corrective Maintenance :

يحدث هذا النوع من الصيانة نتيجة وقوع أخطاء أثناء تشغيل المنظومة على أرض الواقع، وبالتالي يستدعي تصحيح هذه الأخطاء التي لم يتم اكتشافها أثناء إجراء الاختبارات عليها سابقا.

ويحدث العديد من هذه الأخطاء البسيطة التي تسبب في إحداثها المستخدم ويمكنه تصحيحها في حالة تدريبه عليها سابقا.

أمّا الأخطاء والأعطاب التي تتعلق بشفرة البرنامج أو جهاز الحاسوب فهي على عاتق معد المنظومة في حالة وجود ضمانة لمدة معينة أو عقد صيانة.

وهناك ثلاثة أنواع من الأخطاء لها علاقة إمّا بالشفرة أو التوثيق حسب الآتي:

1- أخطاء في شفرة البرنامج: ويعتبر الأقل تكلفة لتصحيحه

2- أخطاء في التصميم: ويعتبر إلى حد ما أكثر تكلفة.

3- أخطاء في المتطلبات: وهي الأكثر تكلفة على الإطلاق في تصحيحها لأنها تتطلب إعادة في التصميم من جديد.

■ صيانة لتحسين المنظومة Perfective Maintenance

حيث يقوم معد المنظومة بإجراء تغييرات للمنظومة بقصد تحسينها لغرض إضافة ميزات جديدة للمنظومة، مثل استخدام الكاميرا الرقمية لأخذ صورة فورية لزبون المصرف في قسم الحسابات الجارية بدلا من إلزام الزبون بجلب صورة من عنده.

■ صيانة للتكييف مع البيئة :Adaptive maintenance

ويعتبر هذا النوع من الصيانة ضروريا، لأن المنظومة يجب أن تعمل بكفاءة وأن تتكيف مع البيئة المحيطة بها التي تتغير باستمرار. لهذا وجب مواكبة هذا التغيير.

وكمثال على ذلك: تغيير لغة برمجة المنظومة من بيسك المرئي القديم (الإصدار الرابع) إلى بيسك المرئي الجديد (الإصدار السادس) .

■ صيانة المنع Preventive Maintenance

وفيها يتم إعادة الهندسة Re-engineering أي إعادة تصميم وتنفيذ المنظومة بقصد منع حدوث مشاكل Prevention أو الإقلال من الحاجة للصيانة.

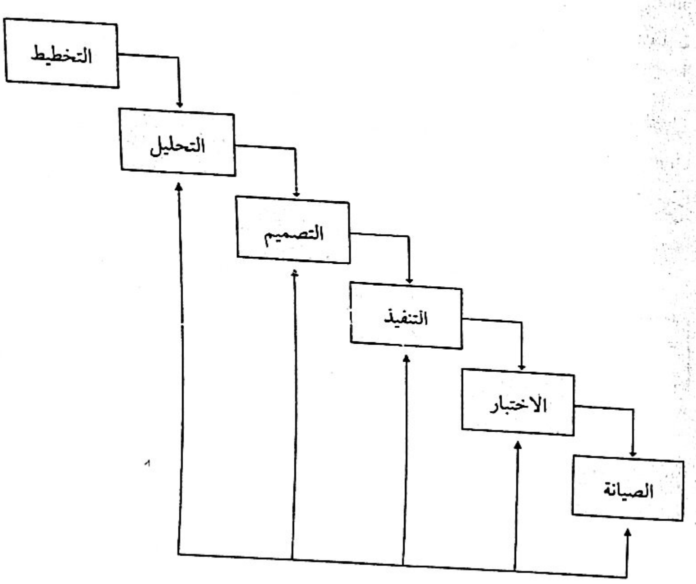
فمثلا إذا تمت كتابة الشفرة بتراكيب يستخدم فيها جمل GO To التي تجعل من الشفرة متشابكة، فيمكن استبدالها بتراكيب أخرى باستخدام مفاهيم البرمجة الهيكلية التي تتجنب هذه الجملة وبالتالي التقليل من الحاجة لعملية الصيانة.

هل هناك نماذج أفضل من النموذج التدفقي؟

إن هذا النموذجWaterfall Model وكما أشرنا سابقا متداول ومستخدم في أغلب المشاريع منذ بداية السبعينيات.

أمّا الآن فأصبح هذا النموذج لايواكب تعقد وتنوع المنظومات خصوصا بعد ظهور الشبكات والانترنت والوسائط المتعددة ونظم قواعد البيانات.

والأهم من ذلك لم يحل مشكلة المتطلبات التي في أغلب الأحيان لا تكون دقيقة ولا كاملة لهذا تم تعديل هذا النموذج بحيث يمكن الرجوع إلى المراحل السابقة في حالة تعديل أو حدوث تغيير في المتطلبات كما في الشكل التالي:



الشكل (8.2.1) النموذج التدفقي المعدّل

وهذا النموذج المعدل هو أيضا لم يحقق الغرض المطلوب. لهذا أصبحت الحاجة لاستخدام نماذج أكثر تطورا ضرورة حتمية، وسارع المختصون بهندسة البرمجيات إلى البحث عن حل بديل وذلك بإعداد نماذج أخرى أكثر تطورا وتراعي مشكلة عدم اكتمال ودقة المتطلبات.

8.3 نموذج العرض التجريبي Prototyping Model

جاء نموذج العرض التجريبي Prototyping Model كمحاولة لحل قصور (عدم نجاح) النموذج التدفقي في حل مشكلة المتطلبات الخاصة بالزبون.

ويقوم معد المنظومة بإعداده ثم بعد ذلك يعرض منظومة تجريبية (نموذج تجريبي pilot system) كعينة sample على الزبون (والمستخدم) يمثل أو يشابه المنظومة النهائية المطلوبة ولكن بصورة أصغر (أي صورة مبدئية) ومدة قصيرة في الإعداد.

وتستخدم هذه العينة عادة إما لتوضيح المتطلبات أو كبداية لإصدار عدة نسخ متتالية للوصول للمنظومة النهائية التي ستسلم للزبون فيما بعد.

وفي هذا النموذج يتم عرض وظائف محددة من المنظومة على الزبون في أقرب وقت ممكن، وإن كان ذلك في مستوى منخفض من الجودة.

والفكرة هنا هي جعل الزبون يرى عينة من المنظومة النهائية في مرحلة مبكرة.

ملاحظة: النموذج التجريبي التطوري Evolutionary Prototype قد يتطور من نسخة إلى أخرى إلى أن يصبح المنظومة النهائية ـ لكن قليلا ما يحدث ذلك.

لماذا نحتاج إلى العرض المبدئي عند إعداد برمجيات جديدة

متى نستخدم العرض التجريبي؟

يستخدم العرض التجريبي Prototype للأسباب الآتية:

1. لتوضيح وتحديد وتنقيح المتطلبات.
2. إقناع الزبون بالجدوى الفنية عندما تكون المخاطرة الفنية عالية.
3. محاولة فهم المسألة المراد حلها قبل الشروع في رصد الموارد والمستلزمات.
4. نسخة بدائية للتوصل إلى النسخة النهائية أو ما يسمى بالإصدار (المنظومة النهائية).

ما الفرق بين مصطلح النسخة والإصدار

والميزة هنا أن الزبون يعطى فرصة لتغيير متطلباته في مرحلة مبكرة من المشروع حيث تعتبر تكلفة التغيير قليلة نسبيا.

وتستخدم طريقة العرض المبدئي كحل لأهم مشكلة في النموذج التدفقي(Waterfall model) ألا وهي مشكلة عدم اكتمال المتطلبات.

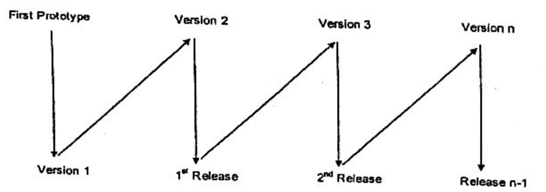
أو يصبح العرض التجريبي هو البداية في إعداد المنظومة حيث يتم تجهيز أول عرض تجريبي وعرضه على الزبون والمستخدم وأخذ الملاحظات والتغيرات بشأنه ليتم بعد ذلك تجهيز أول نسخة first version لعرضها.

وتبدأ عملية إعداد المنظومة لتشمل الملاحظات السالفة الذكر حيث يتم مرة أخرى عرض النسخة الثانية ليتم أخذ المزيد من الملاحظات والإضافات ليتم تعديل المنظومة.

وتصبح النسخة الثانيةsecond version بعد التعديل جاهزة للتحميل على أجهزة الزبون وتسمى في هذه الحالة بالإصدار الأول First release وتستمر عملية عمل النسخ وتحميل الإصدارات إلى أن يتم تجهيز المنظومة بالكامل وينتهي عمل آخر إصدار وبذلك تسلم إلى الزبون نهائيا.

والهدف النهائي منه هو الوصول إلى المنظومة النهائية ابتداء من نموذج تجريبي محدود مبين بالشكل (8.3.1)

ملاحظة: تتم متابعة نموذج العرض التجريبي ابتداءا من اليسار إلى اليمين.



الشكل (8.3.1) نموذج العرض التجريبي (التطوري)

صف أهم نشاطات نموذج العرض التجريبي

ويمكن تلخيص نشاطات نموذج العرض التجريبي Activities في الخطوات التالية:

1. يتم التعرف على المتطلبات من قبل معد المنظومة وبمساعدة المستخدم.
2. يتم تصميم وتنفيذ منظومة عرض مبدئي تقوم بأداء وظائف محدودة كعينة (limited functions).

ويمكن هنا استعمال لغة برمجة مرئية مثل فيجوال بيسك ولغة قاعدة بيانات مثل أكسس نظرا لسهولة برمجتها.

3. تتم تجربة منظومة العرض التجريبي من قبل المستخدم بحضور معد المنظومة ثم بمفرده في موقع الزبون.
4. يعطى الزبون (المستخدم) فرصة استعمال المنظومة التجريبية Pilot system ونتوقع منه في المقابل أن يعطي ملاحظات (تغييرات) حول المنظومة.
5. من التغييرات المطلوبة في الخطوة الرابعة أنه يمكن لمعد البرمجيات أن يعدّل المنظومة ويجهز النسخة اللاحقة.
6. يتم تكرار الخطوات الثالثة والرابعة والخامسة حتى يتم تحقيق متطلبات المستخدم تماما وبالتالي تصبح المنظومة جاهزة للاستخدام.

مزايا العرض التجريبي :

1. يقلل من مشاكل تحديد المتطلبات.
2. يوفر مشاركة ودعم أكبر من طرف الزبون في المراحل المبكرة من المشروع.
3. يقلل من مجازفة تصحيح الأخطاء في المراحل المتأخرة.
4. يمكن استعماله كقاعدة لكتابة المواصفات بجودة أعلى.
5. يمكن استعماله لتدريب المستخدمين قبل تسليم المنظومة.
6. يقلل من وقت إعداد المنظومة.

7. يقلل من التكلفة الإجمالية لإعداد المنظومة.
8. يقلل من مخاطر المشروع
9. تحسين عملية التواصل بين المستخدم ومعد المنظومة.

عيوب العرض التجريبي :

نورد فيما يلي أهم العيوب للعرض التجريبي:

1. أحيانا يكون من الصعب إقناع الزبون بأن المنتوج المبدئي هو للعرض فقط (نظام تجريبي)، وقد يطلب الإسراع في استلام المنتوج، ونتيجة لذلك يستلم منظومة غير مكتملة الجودة لتبدأ مرحلة الصيانة مبكرا.
2. لإعداد منظومة تجريبية سريعة، قد يستخدم المبرمج خوارزميات قليلة الكفاءة ويستخدم أدوات بسيطة في التحليل والتصميم والبرمجة، وهذه الأدوات عادة ما تصبح هي الأدوات الأساسية في إعداد المنتوج النهائي مما قد يؤدي إلى إعداد منتوج قليل الجودة.
3. تخلق إحساسا خادعا من التفاؤل (بأن المنظومة قاب قوسين أو أدنى من الانتهاء من إعدادها).
4. قلة استخدام التوثيق.

يقوم النموذج التزايدي باختصار الوقت اللازم لإعداد البرمجيات وضح هذه العبارة

8.4 النموذج التزايدي Incremental Model

إن الهدف من النموذج التزايدي هو تقليص وقت انتظار الزبون للحصول على المنظومة النهائية خصوصا المعقدة والكبيرة التي تستغرق مدة طويلة (قد تستغرق سنوات) لإعدادها بالكامل. والحل يكمن في تسليم المنظومة إلى الزبون على دفعات، وذلك بتقسيم المشروع (المنظومة) إلى أجزاء يتم تسليمها للزبون على أجزاء تباعا في فترات زمنية محددة ومتلاحقة.

بمعنى أن يتم تقسيم المشروع إلى مجموعة مشاريع جزئية حيث كل مشروع ينهي جزءا من المشروع الكلي (وتمر عليه جميع المراحل) ليتم تسليمه وتشغيله فيما بعد في موقع الزبون.

ثم يتم إعداد المشروع الجزئي التالي وهكذا إلى أن يتم تسليم الجزء الأخير من المشروع.

فمثلا يمكن تقسيم المنظومة المصرفية إلى عدة أجزاء: الحسابات الجارية وحسابات التوفير والحوالات والصرف الأجنبي.

فيتم أولا إعداد منظومة الحسابات الجارية وتسليمها للزبون ومن ثمّ البدء في إعداد الجزء الثاني وهو حسابات التوفير، وهكذا الى أن يتم تنفيذ كل الأجزاء الخاصة بالمنظومة المصرفية بالكامل.

وهناك مثال آخر هو منظومة المبيعات حيث يتم إعداد وتسليم الجزء الخاص بعمل الطلبيات أولا ثم يتم تسليم إعداد فواتير الشراء فى الجزء الثاني بعد فترة زمنية وهكذا.

يبدأ هذا النموذج بتحديد الأهداف وتحويلها إلى متطلبات ليتم تصميمها وتنفيذها واختبارها ثم تسليمها للزبون من أجل البدء في تشغيلها على أرض الواقع.

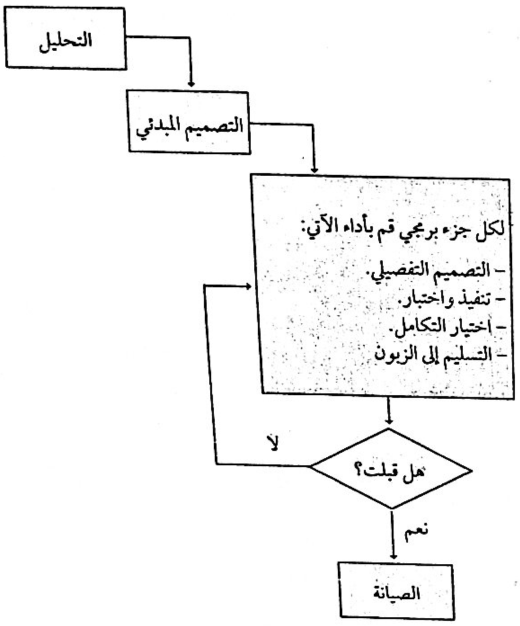
هذا النموذج – كما هو مبين بالشكل (8.4.1) هو عبارة عن خليط من النموذج التدفقي ونموذج العرض التجريبي (waterfall + prototype).

ويستخدم إذا كانت هناك مخاطرة في تنفيذ المنظومة بالكامل دفعة واحدة، أي يستخدم هذا النموذج لتقليل هذه المخاطرة.

خصائص النموذج التزايدي:

- يتكون المنتوج عادة من عدة أجزاء أو بناء Build
- كل جزء (بناء) يتم تصميمه وكتابته واختباره ثم تسليمه للزبون.
- يتكون البناء من قطع برمجية تتفاعل لتعطي وظيفة معينة.
- يعطى المنتوج للزبون تدريجيا (على دفعات) جزءا بعد الآخر.
- بخلاف نموذج العرض المبدئي، يعتبر كل جزء في هذا النموذج منتوجا (اصدار Release) قابلا للعمل في موقع الزبون.

والشكل (8.4.1) يبين هذا النموذج:



الشكل (8.4.1) النموذج التزايدي

مزايا النموذج التزايدي:

1. لا يحتاج إلى الكثير من أعضاء فريق العمل مما يؤدي إلى توفير التكلفة.
2. يؤدي إلى اتصال جيد بين أعضاء الفريق.
3. يستطيع الزبون أن يرى منظومة قابلة للعمل مبكرا.

4. يقلل خطر المجازفة وذلك باستخدام العرض المبدئي.
5. يمكنك تسليم الأجزاء البرمجية العاجلة، وتأجيل الباقي فيما بعد.
6. يستفاد من الخبرة التي تمّ الحصول عليها في إعداد الأجزاء السابقة عند بناء الأجزاء التالية.

8.5 النموذج اللولبي Spiral Model

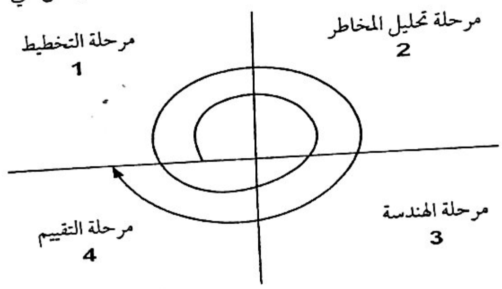
يعتبر النموذج اللولبي Spiral Model خليطا بين النموذج التدفقي والعرض التجريبي. وهذا النموذج يشابه إلى حد بعيد النموذج التزايدي.

وفي هذا النموذج يتم التعامل مع المخاطرRisks التي تهدد المشروع في دورة معينة من هذا اللولب قبل البدء في الدورة التالية من اللولب وهكذا.

ويستخدم هذا النموذج إذا لم توجد ضمانة (وجود مخاطر) كاملة في تنفيذ ونجاح المشروع، ووجود تغيير واضح في المتطلبات حيث يتم التعامل مع هذه المشكلة بواسطة العرض التجريبي. وباستعمال العرض المبدئي تتم مشاركة المستخدم في كل مرحلة من مراحل إعداد المنظومة.

وهذا النوع من النماذج ثبت أنه يلائم المشاريع الكبيرة بصورة خاصة.

وكما هو مبين بالشكل 8.5.1 يتكون النموذج اللولبي من أربعة مراحل هي:



الشكل (8.5.1) النموذج اللولبي

اذكر جميع مراحل النموذج اللولبي مع بعض التفاصيل

مرحلة التخطيط	Planning Phase
مرحلة تحليل المخاطر	Risk Analysis Phase
مرحلة الهندسة	Engineering Phase
مرحلة تقييم الزبون	Customer Evaluation Phase

وفيما يأتي تفاصيل لهذه المراحل:

مرحلة التخطيط Planning Phase

في هذه المرحلة نقوم بالنشاطات التالية:

عرّف المشكلة	Define the problem
حدد الأهداف	Determine objectives
حدد القيود	Determine constraints
عرّف المواصفات	Define specifications
عرّف البدائل	Determine alternatives

مرحلة تحليل المخاطرة Risk Analysis Phase

يجب أن نقوم هنا بنشاطين: نموذج تجريبي لحل مشكلة عدم التأكد من المتطلبات، وأيضا التعرف على المخاطر والحلول البديلة وتحليلها.

مرحلة الهندسة Engineering Phase

الهدف الرئيسي من هذه المرحلة هو إعداد الإصدار الموالي من المنظومة.

وفي هذه المرحلة يجب إنجاز النشاطات التالية:

التصميم المفصل	Detailed design
التشفير	Coding
اختبار الوحدة	Unit testing
اختبار التكامل	Integration testing

مرحلة التقييم من الزبون Customer Evaluation Phase

يتم في هذه المرحلة تقييم المنظومة من حيث سهولة الاستعمال للحصول على النتائج الصحيحة. فيقيّم الزبون المنظومة ويعطي ملاحظات تؤخذ في الاعتبار في الجولة الموالية من التخطيط.

مزايا النموذج اللولبي:

- يستطيع المستخدم أن يقيّم ويغير المتطلبات في مرحلة مبكرة من المشروع.
- مناسب للمشاريع الكبيرة والمعقدة.
- كل من المستخدم ومعد المنظومة يستجيب لأنواع مختلفة من المخاطر في كل تركيبة (دورة في اللولب) وعليه فإن المخاطر ستتناقص شيئا فشيئا.

8.6 نموذج طريقة الجيل الرابع 4GT Model

يستخدم نموذج طريقة الجيل الرابع Fourth Generation Techniques أدوات الجيل الرابع لإعداد المنظومات.

وفكرة هذا النموذج هي تمكين مهندس البرمجيات من تحديد بعض الخصائص للمنظومة وبناء على ذلك يتم تكوين شفرة المصدر تلقائيا.

الأدوات المستخدمة :

1. مكوّن شاشات Screen Generators
2. مكوّن تقارير Report Generator
3. مكوّن شفرة Code Generators
4. لغات قواعد بيانات DMBS
5. لغات شيئية Object _Oriented Languages
6. محرر رسومات Graphic Editors

7. محرر نص Text Editors
8. جدوال الكترونية Spread Sheets

مراحل الجيل الرابع:

اذكر جميع مراحل نموذج الجيل الرابع

يتكون نموذج الجيل الرابع من مجموعة من المراحل وهي :

- مرحلة تحديد المتطلبات Requirements phase

في هذه المرحلة يتم تحديد المتطلبات عن طريق معد البرمجيات وبمساعدة الزبون والمستخدم، ويستعمل في هذا الإطار العرض التجريبي Prototype لعرضه على الزبون والمستخدم كطريقة لتعريف وتنقيح المتطلبات.

- مرحلة التصميم Design phase

هنا نقوم بوضع استراتيجية تصميم المنظومة خاصة للتطبيقات الضخمة، لأن استعمال أدوات الجيل الرابع بدون التصميم الجيد يمكن أن يسبب في مشاكل تؤدي إلى منظومة رديئة الجودة وصعبة القبول من المستخدم.

أما بالنسبة للتطبيقات البسيطة فيمكن الانتقال مباشرة من مرحلة تحديد المتطلبات إلى مرحلة التنفيذ بدون المرور إلى مرحلة التصميم، وذلك باستعمال اللغات غير الإجرائية مثل لغات قواعد البيانات: SQL وOracle .

- مرحلة التنفيذ Implementation phase

التنفيذ باستخدام لغات الجيل الرابع يمكّن معدي البرمجيات من تحديد المخرجات وتركيب البيانات مما قد يسبب في تكوين تلقائي لشفرة البرنامج (باستعمال أدوات تكوين الشفرة).

- مرحلة الاختبار Testing phase

من البديهي أننا نحتاج إلى اختبار مفصل حتى نحصل على منتوج يحقق متطلبات المستخدم، ويشتمل هذا الاختبار على اختبار الأخصائي، واختبار المستخدم.

قارن بين مزايا وعيوب نموذج الجيل الرابع

مزايا نموذج الجيل الرابع:

1. تقليص الوقت اللازم لإعداد البرمجيات (المنظومات).
2. تحسن من عملية إنتاجية البرمجيات.
3. رفع جودة المنتوج البرمجي.

العيوب أو الصعوبات في استخدام النموذج تتمثل في أن:

- أدوات 4GL صعبة الاستعمال على المبرمجين الذين تعودوا على استعمال لغات البرمجة القديمة.
- كفاءة شفرة المصدر الناتجة تلقائيا من هذه الأدوات code generator ليست في المستوى المطلوب وتحتاج الى تعديل.
- استخدام هذه الأدوات في المنظومات الكبيرة قد يسبب مشاكل في عملية الصيانة.

ملاحظات حول استخدام هذا النموذج :

1. استعمال طرق الجيل الرابع للتطبيقات الكبيرة يتطلب مجهودات أكبر في إعداد البرمجيات من حيث التحليل والتصميم والاختبار.
2. طرق الجيل الرابع قد تكون الأفضل للعديد من التطبيقات خاصة عندما نستعمل أدوات حديثة مثل CASE ومكونات الشفرة.
3. من الإحصائيات التي تم تجميعها بخصوص استعمال 4GT تم استنتاج أن هذه الطريقة تقلّص من وقت إعداد البرمجيات خصوصا للتطبيقات الصغيرة والمتوسطة .
4. إمكانية تقليص الوقت اللازم لتحليل وتصميم التطبيقات الكبيرة.

8.7 نموذج الإعداد السريع Rapid Development Model (RAD)

يستخدم هذا النموذج لإعداد البرمجيات التي تعتبر من النوع الخطي.

كيف تحصل على المنتوج النهائي في نموذج الإعداد السريع

حيث يتم إعداد المنتوج النهائي باستخدام طريقة التنفيذ على دفعات للمنظومة.

والوقت المتوسط لبناء منظومة متكاملة قابلة للتشغيل قد يتطلب مدة من 60 إلى 90 يو...

ويعتمد هذا النموذج على إعادة استعمال المكونات (Component reuse).

8.7.1 مراحل نموذج الإعداد السريع:

اذكر جميع مراحل نموذج الإعداد السريع

تتمثل في المراحل التالية:

1. **مرحلة نمذجة الأعمال Business modeling phase:**

يتم هنا تعريف انسياب البيانات بين مختلف وظائف النظام.

2. **مرحلة نمذجة البيانات Data modeling phase:**

في هذه المرحلة يتم تعريف الكائنات وخصائص كل منها والعلاقات بينها.

3. **مرحلة نمذجة العمليات Process modeling phase:**

يتم هنا تحديد العمليات على كل كائن (تم تعريفه في مرحلة نمذجة البيانات).

قد تشمل هذه العمليات الإضافة والإلغاء والتعديل واسترجاع كائن بيانات.

4. **مرحلة تكوين التطبيق Application generation phase:**

تستعمل هذه المرحلة (مرحلة البرمجة) لغة الجيل الرابع كأداة لبناء البرمجيات.

5. **مرحلة الاختبار Testing phase**

تعتبر عملية الاختبار عملية بسيطة لأنه كما ذكرنا سابقا يركز هذا النموذج على إعادة استعمال الأجزاء وهو مفهوم ذو علاقة بالبرمجة الشيئية مما يمكننا من الاستفادة من الأجزاء التي تم اختبارها في منظومات سابقة.

قارن بين مزايا وعيوب نموذج الإعداد السريع

8.7.2 مزايا نموذج الإعداد السريع

1. يمكن استعماله في جميع أنواع التطبيقات بشرط أن يكون المنظومة المقترحة مقسمة إلى أجزاء مناسبة.

2. يتعامل مع التقنية الشيئية Object Technology التي تركز على إعادة استعمال الأجزاء.
3. يستغرق وقتا أقل في إعداد البرمجيات من معظم النماذج الأخرى.
4. باستخدام RAD نحصل على منظومات ذات جودة عالية.
5. يزيد من إنتاجية المبرمجين.
6. المنظومات المنفذة باستخدام هذا النموذج ذات تكلفة منخفضة.

8.7.3 عيوب نموذج الإعداد السريع:

7. يحتاج إلى عدد أكبر من المبرمجين (حتى نشكل فرق عمل RAD) وذلك بسبب أن كل وظيفة أو جزء يتم إعداده بواسطة فريق منفصل.
8. يحتاج إلى مشاركة وتعاون كامل بين الزبون ومعد البرمجية وإلاّ فإن المشروع سيفشل.

8.8 أسئلة للمراجعة

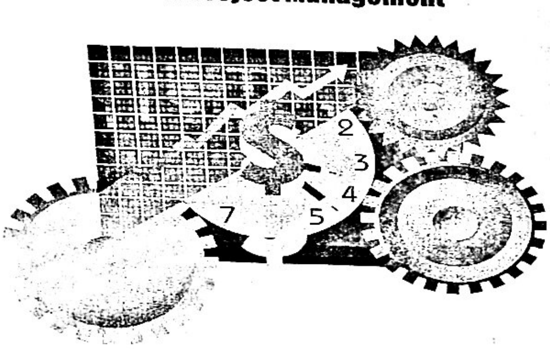
1. لماذا نحتاج إلى العرض المبدئي عند إعداد برمجيات جديدة؟
2. ما الفرق بين المصطلحين «نسخة» و«إصدار» في نموذج العرض المبدئي
3. صف أهم نشاطات تنجز في نموذج العرض المبدئي
4. يقوم النموذج التزايدي باختصار الوقت اللازم لإعداد البرمجيات. وضح هذه العبارة.
5. اذكر بعض الأدوات المستخدمة في نموذج طرق الجيل الرابع.
6. اذكر جميع مراحل نموذج طرق الجيل الرابع.
7. قارن بين مزايا وعيوب نموذج طرق الجيل الرابع.
8. كيف نحصل على المنتوج النهائي في نموذج الإعداد السريع؟

9. اذكر جميع مراحل نموذج الإعداد السريع.
10. قارن بين مزايا وعيوب نموذج الإعداد السريع.
11. اذكر مراحل النموذج اللولبي مع بعض التفاصيل عن كل مرحلة
12. اذكر مزايا وعيوب النموذج اللولبي

الفصل التاسع

إدارة مشاريع البرمجيات

Software Project Management



1.1 مقدمة

لقد أدى فشل الكثير من المشاريع البرمجية Software projects في الستينيات والسبعينيات من القرن العشرين إلى ظهور دراسة علم إدارة المشاريع Project management كمجال حيوي له علاقة مباشرة بهندسة البرمجيات من أجل إعداد منظومات برمجية في صورة مشابهة للمشاريع الهندسية الأخرى كالمعمار والصناعات الميكانيكية وغيرها.

لماذا يجب على مهندس البرمجيات دراسة إدارة المشاريع

ومن أجل ذلك نجد أن علم إدارة المشاريع يعتبر من المقررات المهمة في كليات الهندسة وترجع أهمية إدارة المشاريع في إعداد المنظومات (البرمجيات) إلى الأسباب التالية:

1. نجاح أي مشروع برمجي Software project يتطلب تخطيطا Planning ومتابعة جيدة Monitoring وتوثيقا Documentaion يبين مدى تقدم إنجازه، وهذه بالطبع مصطلحات تتعلق بعلم إدارة المشاريع يجب فهمها والعمل بها.
2. يطمح معظم مهندسي البرمجيات أن يصبحوا مديري مشاريع Project managers، لهذا وجب عليهم الإلمام بهذا المجال.
3. تدار المشاريع الكبيرة عادة من قبل مجموعة من الأفراد (أعضاء فريق إعداد المنظومة) الذين يحتاجون بالطبع إلى وظيفة التوجيه Directing للوصول إلى نهاية مثمرة للمشروع. وتتعلق هذه الوظيفة بإدارة المشاريع.
4. التكلفة الزائدة للمشاريع المتأخرة سببت ما يعرف بأزمة البرمجيات، ويمكن لإدارة المشاريع أن تساهم في حل هذه الأزمة من خلال تقنيات وطرق تقدير التكلفة.
5. إدارة المشاريع هي مركز نشاط إعداد أي مشروع برمجي، لأن كل مرحلة من إعداد المشروع تتطلب تخطيطا ومتابعة وتوجيها وكل هذه الوظائف تتعلق بعلم إدارة المشاريع.

9.2 ما معنى إدارة المشاريع؟ What is Project Management?

ماهو المشروع

المشروع Project هو عبارة عن مجموعة نشاطات Activities تربطهم علاقة للوصول لهدف أو غاية ما.

وللمشروع مجموعة خصائص نذكر منها:

نقطة بداية	Starting point
نقطة توقف	Stopping point
ميزانية	Budget
شخص مسؤول	Responsible person
مجموعة من المتطلبات	Set of requirements
أهداف	Goals
نطاق أو حدود	Scope
قيود	Constraints
موارد أو مستلزمات	Resources

أمّا إدارة المشاريع البرمجية Software Project Management فهي مجموعة من الأدوات tools والقواعد rules والأعراف conventions والممارسات practices والطرق methods والمعرفة knowledge التي توفر التخطيط Planning والمتابعة controlling والتوجيه directing والتنظيم Organizing والتوظيف Staffing اللازم لإدارة مشروع برمجي والوصول به إلى نهاية ناجحة.

ويمكن تعريفه أيضا بأنه دراسة واستخدام الأدوات والطرق والممارسات اللازمة في التخطيط والمتابعة والتنظيم والتوجيه والتوظيف لإدارة مشروع برمجي للوصول إلى نهاية ناجحة.

ماهو الهدف الرئيسي من إدارة المشاريع البرمجية؟

أما الغرض الرئيسي من استخدام علم إدارة المشروع لإعداد المنظومات البرمجية فيتمثل في التحكم في نطاق Scope ووقت Time وتكلفة Cost وجودة Quality المشروع البرمجي.

ما هو المشروع البرمجي الناجح؟ الخصائص التي يجب أن يتمتع بها المشروع

لكي يكون المشروع البرمجي ناجحا وبأقل تكلفة يجب أن يكون له:

1. جدول زمني محدد.
2. ميزانية مخصصة.
3. موارد كافية.
4. تخطيط جيد.
5. متطلبات كافية.
6. مراجعة كافية.
7. توجيه جيد.
8. متابعة جيدة لسير تنفيذه.
9. تنظيم كفء.
10. قيادة حكيمة وقادرة.

9.3 مدير المشروع Project Manager

تعريف أكمل

هو الشخص المسؤول عن نجاح المشروع وله كامل الصلاحيات والمسؤوليات، ويتحمل أية تبعات تؤثر في نجاح أو فشل المشروع.

ماهي الخصائص التي يجب أن يمتلكها مدير المشروع

ويجب أن يملك مدير المشروع الخصائص التالية:

قيادة الفريق	Team leadership
متحدثا رسميا	Spokesman
متابع	Monitor
مفاوضا	Negotiator

مخصص ميزانية	Budget allocation
حلاّل المشاكل	Problem solver

ويواجه مدير المشروع أثناء ممارسة عمله في مراحل المشروع المختلفة بعض المشاكل التي عليه الانتباه إليها ومحاولة تفاديها أو التقليل منها ما أمكن.

ومن هذه المشاكل ما يلي:

1. عدم دراية الزبائن والمستخدمين بمجال وتقنيات الحاسوب.
2. تعرض المنظومات تحت الإنجاز لعملية التغيير المستمر في المتطلبات، سواء أثناء إعداد هذه المنظومات أو بعد تسليمها للزبون لتشغيلها على أرض الواقع
3. تعرض المشروع لبعض المخاطر المختلفة.
4. صعوبة التفاهم مع المستخدم.

مهام مدير المشروع:

يتحمل مدير المشروع كافة المسؤوليات والمهام عندما يتم تكليفه بمهمة قيادة فريق المشروع.

والنقاط التالية تبين أهم النشاطات التي يجب عليه أداؤها:

1. كتابة مقترحات Proposals لإعداد برمجيات جديدة.
2. إعداد جدول المشروع وتقدير مدته التقديرية.
3. عمل جلسات المراجعة المختلفة مع الزبون والمستخدم وأعضاء فريق المنظومة.
4. تخطيط المشاريع.
5. اختيار أعضاء المشروع.
6. كتابة التقارير.
7. تقديم العروض الشفوية عن المشروع.

8. التواصل مع الزبون والمستخدم
9. التواصل مع أعضاء فريق المشروع

ملاحظة: يتم اختيار مدراء المشاريع عادة من محللي نظم ذوي خبرة.

9.4 لماذا تفشل المشاريع؟

تفشل المشاريع البرمجية بسبب الآتي:

1. ضعف الاتصال والتواصل بين الزبون ومدير المشروع.
2. ضعف تعريف الأهداف بالتحديد وبالتالي ضعف المتطلبات، حيث إن هذه الأهدا[ف] ستتحول إلى المتطلبات التي هي أهم عناصر المنظومة.
3. الإدارة السيئة حيث إن مدراء المشاريع يتم اختيارهم إمّا من الإداريين أو من المبرمجين الذين يجهلون كل ما يتعلق بالمشروع وبنوده.
4. عدم التقيد بالمواصفات القياسية للمشروع الناجح.
5. إهمال التوثيق وجعل مرحلة البرمجة أهم المراحل .
6. ضعف التخطيط للمشاريع البرمجية.
7. سوء تقدير الموارد.

طرق حل مشاكل إدارة المشاريع:

لتجنب العديد من المشاكل التي قد تتم مواجهتها يجب أن:

- يتم اختيار مديري المشاريع على أساس قدرتهم في إدارة المشاريع وليس على أساس قدرتهم الفنية كمبرمجين فقط .
- يعدّ مدير المشروع تقديرا دقيقا للتكاليف والوقت اللازم للمشروع.

- يتم التعامل مع التوثيق Documentation كجزء مهم في المشروع.
- يتم تعليم وتدريب الأفراد للحصول على المهارات الخاصة بإعداد البرمجيات.
- يتم تعريف وتحديد صفات الجودة Quality attributes.

وظائف إدارة المشاريع: Project management functions

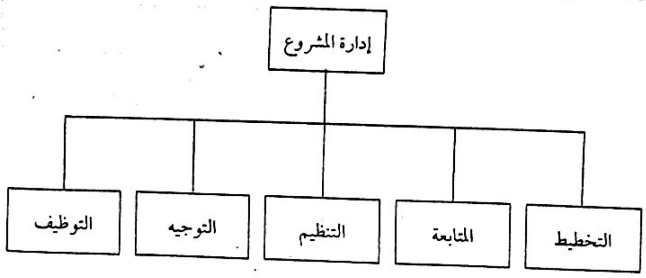
ماهي الوظائف الرئيسية لادارة المشاريع

إن أهم الوظائف التي تقوم بها إدارة المشاريع (وهي إلى حد ما تشبه الوظائف، الخاصة بالإدارة العامة) هي:

التخطيط	Planning
التوجيه	Directing
المتابعة	Controlling
التنظيم	Organizing
التوظيف	Staffing

ويتم تنفيذ هذه الوظائف باستخدام أدوات وفنيات وطرق خاصة.

ويقوم مدير المشروع بدور مهم في تنفيذ هذه الوظائف (خاصة التخطيط) والشكل (9.4.1) يبين هذه الوظائف:



الشكل (9.4.1) وظائف إدارة المشاريع البرمجية

9.5 التخطيط Planning

التخطيط هو كل النشاطات الإدارية التي تجرى لغرض تحديد الأهداف لمشروع ما والطرق أو السبل (خطة المشروع) اللازمة لتحقيق هذه الأهداف.

نشاطات التخطيط: Planning activities

تشمل وظيفة التخطيط في المشروع البرمجي أداء النشاطات والمهام الآتية:

1. تحديد أهداف المشروع objectives.
2. تحديد مستلزمات المشروع resources.
3. تقدير تكاليف المشروع البرمجي cost.
4. تحديد نطاق المشروع Scope.
5. تحديد خصائص الجودة quality attributes المطلوبة من المنظومة المقترحة، مثل الاعتمادية وسهولة الاستعمال والصيانة... إلخ.
6. وضع الجدول الزمني للمشروع البرمجي Schedule.
7. إعداد خطة خاصة لتوفير العاملين بالمشروع.
8. وضع المعايير والسياسات Standards.
9. تحديد أدوات التحليل Analysis tools used.
10. تحديد أدوات التصميم Design tools used.
11. تحديد أدوات البرمجة Programming languages used.
12. تحديد أنواع الاختبار Types of tests.
13. تحديد التوثيق Documentation الذي سوف يتم إعداده في المشروع.
14. تحديد المنتوجات التي ستسلم للزبون (Deliverables).

9.6 المتابعة Controlling

المتابعة Controlling أو المراقبة Monitoring هي جميع النشاطات الإدارية التي تنفذ للتأكد من أن العمليات الفعلية تتم طبقا للخطة المرسومة.

نشاطات المتابعة :

بعد عملية التخطيط وعند البدء في تنفيذ المشروع، يقوم مدير المشروع بأداء الوظائف الآتية:

1. إنشاء نظام تقارير Reporting system خاص بالمشروع.
2. تحديد واستعمال أدوات التقارير.
3. تحديد نوع وعدد ومعد التقارير.
4. تحديد تقنيات البرمجة اللازمة للتحكم في الأداء.
5. تحديد معايير الأداء Set performance standatds .
6. تحديد معايير جودة البرمجيات Set Quality Standards.
7. تحديد مقاييس الاعتمادية وقابلية الصيانة.
8. قياس النتائج Measure results.
9. استخدام فنيات لقياس ما إذا كان المنتوج البرمجي يحقق المتطلبات.
10. استعمال المراجعة Review في نهاية كل مرحلة لقياس تقدم إنجاز المشروع.
11. استعمال معايير لقياس إنتاجية المبرمج Programmer productivity.
12. اتخاذ إجراءات تصحيحية corrective actions إذا وجد أي انحراف عن الخطة المرسومة.

عرف المصطلح

9.7 التوجيه Directing

التوجيه هو جميع النشاطات الإدارية التي تمارس لقيادة المرؤوسين بكفاءة وتأثير لغرض تحقيق الأهداف وذلك باستعمال التحفيز والاتصال والقيادة.

نشاطات التوجيه:

ماهي النشاطات

ولكي يصل مدير المشروع إلى إنهاء المشروع بالصورة المرضية والناجحة عليه أن يقوم بأداء النشاطات الآتية:

1. تحديد المهام والمسؤوليات — Define responsibilities
2. تحديد الصلاحيات (تسلسل الأوامر) — Define Authorities
3. حل مشاكل النزاع — Resolve conflicts
4. تشجيع التفكير المستقل والإبداع — Creativity and innovation
5. تنسيق أنشطة المشروع من خلال التواصل — Coordinate Activities
6. تحفيز ورفع الروح المعنوية للعاملين بالمشروع — Inspire

9.8 التنظيم Organizing

التنظيم Organizing هو كل النشاطات الإدارية التي ينتج عنها تصميم هيكلية للمشروع توضح كلا من المهام والصلاحيات لكل فرد عامل في هذا المشروع.

وأهم نشاطات التنظيم هي: اذكر بعض نشاطات التنظيم

- اختيار نوع الهيكل التنظيمي للمشروع (الوظيفي والمصفوفة والمشروع).
- تحديد نطاق ودرجة الاختصاص والصلاحيات لمدير المشروع.
- تحديد المهام tasks والصلاحيات لكل فرد في المشروع.
- تحديد الوصف الوظيفي لكل وظيفة تتعلق بالمشروع البرمجي ومسميات الوظائف.
- تحديد المؤهلات المطلوبة لكل وظيفة تخص المشروع.

أنواع هياكل تنظيمات المشروع: ماهي أنواع التنظيم

هناك ثلاثة انواع من هياكل تنظيم المشروع وهي:

1. النوع الأول: هيكلية التنظيم الوظيفي Functional Organization

هذه الهيكلية مبنية على مفهوم المهمة أو مجموعة المهام ذات العلاقة بحيث تكون هذه المهام في تسلسل معين لتؤدى كوظيفة معينة (وظيفة التحليل أو التصميم مثلا)، حيث يقسّم المشروع عادة إلى عدة مراحل، وكل مرحلة(مجموعة مهام) يقوم بأدائها قسم مختص فمثلا: قسم التحليل يقوم بعملية تحليل المنظومة وقسم التصميم هو المسؤول عن التصميم وهكذا أي أن إدارة الحاسب الالى تنقسم إلى عدة أقسام وظيفية Functional departments (قسم التحليل وقسم التصميم وقسم البرمجة...الخ) حيث يبدأ المشروع بأول قسم، وهو قسم التحليل فيصبح المشروع تحت إمرة رئيس قسم التحليل وعند الانتهاء من عملية التحليل يتحول المشروع إلى القسم التالي ألا وهو قسم التصميم أي تحت إمرة رئيس قسم التصميم لتبدأ عملية تصميم المشروع... وهكذا.

ونلاحظ هنا أن المشروع ليس له مدير مشروع واحد بل مسؤولية أدائه تتوزع بين رؤساء الأقسام المختصة. أي أن هناك توزعا في المهام وتوزعا في الصلاحيات.

2. النوع الثاني: هيكلية التنظيم على شكل مشروع Project Organization

هذا التنظيم مبني على فكرة المشروع المعين والمحدد حيث يتم إعطاء مدير المشروع المسؤولية الكاملة والاختصاصات الواسعة والموارد اللازمة لإتمام المشروع. ومسؤولية مدير المشروع هنا تحقيق الأهداف بالموارد المعطاة.

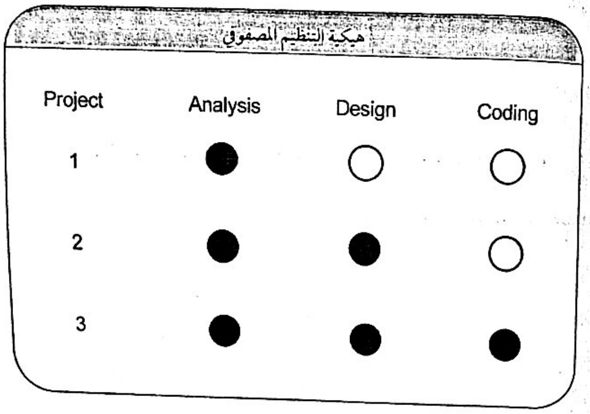
ولأن مدير المشروع في هذا النوع من التنظيم الهيكلي له صلاحيات واسعة في تعيين وفصل وتدريب العاملين بالمشروع (عكس النوع السابق) فإنه يتمتع بمرونة تساعده في تحقيق الأهداف بسرعة ويسر.

3. النوع الثالث: هيكلية التنظيم المصفوفي Matrix Organization

في هذا النوع لا يملك مدير المشروع صلاحيات ومسؤوليات مطلقة لإنهاء المشروع.

حيث تتوزع المهام والمسؤوليات بين مدير المشروع ومدير الموارد(مثل رئيس قسم التحليل ورئيس قسم التصميم ...وهكذا)أي أن هذا النوع هو خليط بين النوع الأول والنوع الثاني.

وفي الواقع يوجد لدينا في التنظيم المصفوفي مديران للمشروع: المدير الأول هو مدير المشروع والمدير الثاني هو رئيس القسم المختص، كما مبين بالشكل (9.8.1)



الشكل (9.8.1) هيكلية التنظيم المصفوفي

في هذا الشكل نجد أن:

في السطر الأول: المشروع رقم 1 يدار بواسطة مديرين: مدير المشروع ورئيس قسم التحليل.

وفي السطر الثاني: يدار المشروع رقم 2 من طرف كل من مدير المشروع ورئيس التصميم.

لاحظ في السطر الثاني أن رئيس قسم التحليل كان قد شارك في إدارة المشروع في مرحلة التحليل قبل أن يصل المشروع إلى القسم التالي ألا وهو قسم التصميم ليتولى رئيس قسمه المشاركة مع مدير المشروع في أداء عملية التصميم وهكذا.

وبصورة عامة نجد أن في أي مرحلة من المشروع يشارك مشرف أو رئيس قسم تلك المرحلة مع مدير المشروع في توجيه وإدارة المشروع.

9.9 مزايا وعيوب هيكليات تنظيمات المشاريع

استعرضنا في البند السابق الأنواع الثلاثة لهيكليات تنظيم المشاريع.
ولكن السؤال هو أي منهم أفضل؟
وقبل الاجابة دعنا نستعرض الميزات والعيوب لكل واحدة.

أولا: هيكلية التنظيم الوظيفي: Functional Organization Structure

في هذه الهيكلية يتم تمرير المشروع على الوحدات الإدارية لقسم الحاسوب ليتم تنفيذ الجزء الخاص بكل وحدة.

ومن المزايا والعيوب نسرد التالي.

المزايا :

1. يبدأ المشروع مبكرا بسبب أن الأقسام الوظيفية مثل قسم التحليل والتصميم والبرمجة اللازمة للمشروع موجودة أصلا .
2. المعايير Standards الخاصة باعداد وتقييم المشروع موجودة أصلا في الأقسام الوظيفية المختصة ولا حاجة لتحديدها
3. يمكن توفير التدريب والتعيين ببساطة.

العيوب:

1. لا يوجد مسؤول بعينه ولا نعلم من هو صاحب الصلاحية بالكامل عن المشروع.
2. صعوبة متابعة المشروع بسبب أن كل قسم مستقل عن القسم الآخر ولا يوجد مدير مشروع بالمعنى المتعارف به.
3. يصلح للمشاريع الصغيرة فقط.
4. صعوبة التواصل بين العاملين بالمشروع في الأقسام المختلفة وذلك لاستقلالية الأقسام عن بعضها.

ثانيا: هيكلية تنظيم المشروع Structure Project Organization

وتعتمد هذه الهيكلية على مدير المشروع الواحد المستقل.

المزايا:

1. لمدير المشروع صلاحية ومسؤولية كاملةعلى المشروع.
2. سرعة اتخاذ القرارات
3. يمكن خلق حوافز قوية للعمل من طرف مدير المشروع.

العيوب:

1. ضرورة تكوين هيكلية جديدة للمشروع.
2. الحاجة لإعداد معايير جديدة للمشروع
3. صعوبة التدريب.

ثالثا: هيكلية التنظيم المصفوفي Matrix Organization Structure

ما هي مزايا وعيوب كل من ...

المزايا:

1. أفضل من التنظيم الوظيفي من حيث الصلاحيات والمسؤوليات.
2. في هيكلية تنظيم المصفوفة التدريب يمكن إجراؤه بصورة أبسط مما لو استعملنا هيكلية تنظيم المشروع.
3. وجود المعايير والمقاييس الخاصة بالمشروع والموجودة في الأقسام الوظيفية المختلفة مسب

العيوب:

1. قد يحدث تضارب لأن الصلاحية والمسؤولية مشتركة بين مديرين: مدير المشرو ورئيس القسم الوظيفي (أو ما يسمى بمدير المستلزمات).
2. ضرورة استخدام التوثيق كوسيلة للتواصل بين المديرين.
3. التنافس على الموارد والمستلزمات بين مديري المشاريع المختلفة.

9.10 عوامل اختيار النوع الأفضل لهيكليات تنظيم المشاريع

تعتمد أفضل هيكلية على عدة عوامل كما يلي:

المعامل	الوظيفية	المصفوفة	المشروع
حجم المشروع	صغير	متوسط	كبير
أهمية المشروع	صغيرة	متوسطة	عالية
المدة	قصيرة	طويلة	متوسطة
درجة التعقيد	قليلة	متوسطة	عالية
التقنية	عادية	معقدة	جديدة

9.11 التوظيف Staffing ماهو التوظيف

التوظيف هو جميع النشاطات الإدارية التي توفر العنصر البشري للهيكل التنظيمي من خلال اختيار مناسب لأعضاء فريق المشروع.

وتتمثل اهمية التوظيف أو التعيين بالأعمال التالية: ما هي النشاطات التي يشملها

1. القيام باختيار وتوظيف النخبة المؤهلة لكل وظيفة شغل في المشروع:
   - مدير المشروع
   - أعضاء فريق المشروع.
2. تعليم وتدريب العنصر البشرى (أعضاء فريق إعداد المنظومة)
3. تحديد المرتبات والمكافآت.

9.12 أسئلة للمراجعة

1. لماذا يجب على مهندس البرمجيات دراسة إدارة المشاريع؟

2. عرّف ما هو المقصود بالمشروع.
3. ما هو الهدف من إدارة المشاريع البرمجية؟
4. ما هي الخواص التي يجب أن يتمتع بها المشروع البرمجى الناجح؟
5. أكمل: مدير المشروع هو الشخص المسئول عن -نجاح-- المشروع ولديه كامل الصلاحيات والمسؤليات- على المشروع بالكامل.
6. ما هي الخصائص التي يجب أن يملكها مدير المشروع؟
7. ما هي المشاكل التي تواجه عادة مدير المشروع؟
8. ما هي النشاطات التي عادة ما يقوم بها مدير المشروع؟
9. اذكر ثلاثة طرق على الأقل لحل مشاكل مدراء المشاريع.
10. ما هي الوظائف الرئيسية لإدارة المشاريع؟
11. عرّف معنى (التخطيط) في إدارة المشاريع البرمجية.
12. ما هي النشاطات التي يمارسها مدير المشروع؟
13. عرّف المتابعة كوظيفة من وظائف إدارة المشاريع؟
14. ما هي النشاطات الرئيسية للمتابعة؟
15. عرف المصطلح (توجيه).
16. ما هي النشاطات الرئيسية للتوجيه؟
17. «تنظيم المشروع ضروري لتحقيق نجاحه». وضح هذه الجملة.
18. اذكر بعض نشاطات التنظيم.
19. ما هي أنواع التنظيم؟
20. قارن بين هيكلية التنظيم الوظيفي وهيكلية تنظيم المشروع من حيث المزايا والعيوب لكل منهما.

21.متى يجب استعمال هيكلية تنظيم المصفوفة؟

22.ما هي مزايا وعيوب كل من :

- Functional organization
- Project organization
- Matrix organization

23.ماهو نوع هيكلية التنظيم الذي تختاره لمشروع يملك:

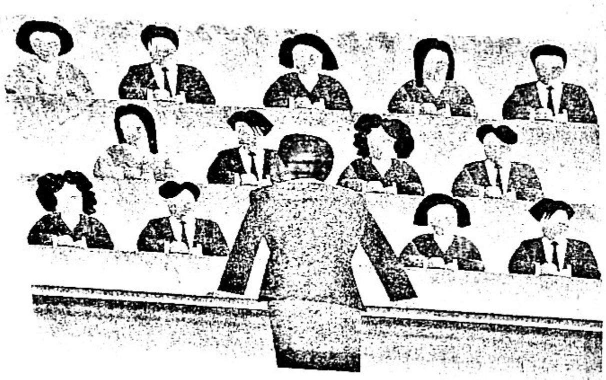
- حجم صغير وتعقيد بسيط.
- حجم كبير وغموض عالي.
- حجم متوسط وتقنية معقدة.

24.ماهو «التوظيف» وما هي النشاطات التي يشير إليها؟

الفصل العاشر

العوامل البشرية وواجهة المستخدم

Human Factors & User Interface



10.1 مقدمة:

كما أسلفنا في الفصول السابقة، كان لظهور أزمة البرمجيات أثر مباشر في بزوغ علم هندسة البرمجيات الذي كان هدفه الأول حل جميع المشكلات التي تمت مواجهتها في فترة الستينيات والتي سميت في تلك الآونة بأزمة البرمجيات.

ولقد حرص العلماء في تلك الآونة على التنقيب في العلوم الأخرى لعلها تساهم في حل هذه الأزمة فاستعان علماء الحاسوب بعلوم الهندسة والاقتصاد وعلم الإدارة وعلم الاجتماع وحتى علم النفس لهذا الأمر. فمثلا ساهم علم الإدارة باستخدام التخطيط السليم في حل مشكلة تأخر تنفيذ المشاريع البرمجية.

أمّا علم النفس Psychology وبكونه علم دراسة سلوك الإنسان فساهم في حل المشكلة بشقيها:

الشق الأول: يتعلق بفريق إعداد المنظومة Development Team:

والمشكلة هنا تتعلق بنقص الإنتاجية Productivity نتيجة لزيادة احتياجات الزبون مما أدى إلى كبر وتعقد تلك المنظومات وبالتالي كانت ضرورة تحسين عملية الاختيار للأفراد وقدرات الأفراد الذين يستخدمون هذه الأدوات.

وفي هذا الفصل سوف ندرس كيف أن الإنسان يتأثر بالجانب النفسي، وهذا الجانب بالطبع يؤثر على الإنتاجية.

فعلم النفس يدرس مثلا ذكاء الإنسان Human Intelligence وسمات سلوكه Personality Traits اللذين يجب على مدير المشروع أن يراعيهما عند توظيف هؤلاء الأفراد للحصول على زيادة في الإنتاجية لدى أفراد المنظومة.

لقد لاحظ علماء النفس أن الإنتاجية تتأثر بعدة عوامل منها الأدوات والتقنيات المستخدمة وأيضا سلوك وقدرات الأفراد.

أما الشق الثاني من الحل فهو يتعلق بالمستخدم User:

أي بعبارة أخرى رضا وقبول المستخدم للمنظومة وتحسين عملية تواصله مع الحاسوب من خلال ما يعرف بواجهة المستخدم User interface.

حيث أن المنظومة سيتم تسليمها للمستخدم وإذا لم يراع الجانب النفسي من حيث معرفة إمكانيات وقدرات ورغبات ونفسيات هؤلاء المستخدمين فلن يتم قبولها.

ومن هذا نستنتج أن علم النفس الذي هو علم سلوك الإنسان يعتبر من الروافد المهمة في علم هندسة البرمجيات، ومن ثمّ فمن الضروري دراسة العوامل البشريةHuman Factors التي تؤثر وتتأثر بالمنظومة.

10.2 دوافع دراسة العوامل البشرية

تعتبر دراسة العوامل البشرية من قبل معد المنظومة أو بصورة أخص مدير المشروع من الأمور المهمة عند إعداد المنظومات لكي:

1. يتفهم مديرو المشاريع البرمجية الموظفين التابعين لهم كأفراد وكبشر (لهم مشاعر وأحاسيس) وأن يأخذوا بهذا الاعتبار عند التعامل معهم.
2. يأخذ المصممون في الاعتبار عند تصميم واجهة المستخدم بأن المنظومة البرمجية سيتم استعمالها بواسطة بشر (مستخدمين)، ومن ثم يجب أن يكون التصميم مراعيا لقدراتهم وإمكانياتهم ورغباتهم، وإلاّ فإن المنظومة لن يتم قبولها واستعمالها.
3. يتم فهم أن إنتاجية المبرمج Programmer productivity تؤثر على تكلفة إعداد المنظومة البرمجية، ومن ثمّ يجب دراسة العوامل البشرية لزيادة الإنتاجية.
4. يتم اختيار رئيس وأعضاء فريق إعداد المنظومة بناء على التوازن بين نوع الشخصية والمهارات التقنية وسنوات الخبرة.

هناك عوامل بشرية مهمة تؤثر بشكل كبير على إنتاجية أعضاء فريق إعداد المنظومة وبالتا ستؤثر على التكاليف وزمن تسليم المنظومة .

ومن هذه العوامل:

10.3 الشخصية البشرية: Human Personality

نظرا لتعقد المنظومات البرمجية فإن إعدادها لايتم من قبل فرد واحد بل يحتاج إلى فريق عمل، وكل فرد من هذا الفريق له شخصيته الخاصة به، بمعنى أن الفريق سيتكون من أفرد مختلفي الشخصية.

وتعتبر المعرفة بأمور الشخصية مهمة وضرورية للغاية نظرا للأسباب الآتية:

1. تختلف نوع الشخصية من شخص لآخر، فهناك أنواع قد تكون مناسبة لعمل البرمجة أي التعامل مع جهاز الحاسوب أغلب الوقت، بينما شخصيات أخرى قد تميل أكثر للتوثيق أي القدرة على الكتابة الفنية، وشخصيات ثالثة قد تميل أكثر للتواصل مع الزبون كمحللي نظم .
2. تعتبر الشخصية البشرية متقلبة ومتغيرة، فبعض الناس قد يكونون هادئين أحيانا، وقد تنتابهم الكآبة والعصبية أحيانا أخرى. وهؤلاء الأفراد الذين سيتم اختيارهم بشر، لهذا يجب مراعاة ذلك من قبل مدير المشروع.
3. تختلف سمات الشخصية Personality traits من شخص لآخر.

لذلك على مدير المشروع أن يراعي هذه السمات عند تشكيل فريق العمل.

نسرد هنا بعض هذه السمات:

Relaxed	مرتاح	Nervous	عصبي
Serious	جدي	Quiet	هادئ
Dominant	قوي التأثير	Talkative	كثير الكلام

موثوق	Reliable	محترس	Careful
نشط	Active	متفائل	Optimistic
متشائم	Pessimistic	عدواني	Aggressive
ذكي	Intelligent	عملي	Practical
حساس	Sensitive	انعزالي	Isolated
اجتماعي	Social	متعاون	Cooperative
مسالم	Peaceful	مزاجي	Moody

فمثلا يجب أن يتصف مدير المشروع البرمجي بالشخصية الاجتماعية (قوة التأثير)، بينما يتصف المبرمج بالذكاء وحب التعاون مع الآخرين.

10.4 الذكاء البشري Human Intelligence

من البديهي أن يختلف الأفراد في ذكائهم intelligence ومهاراتهم skills ومعلوماتهم knowledge عن بعضهم.

وتقع المسؤولية على مدير المشروع Project manager عند اختيار المحللين والمبرمجين والأفراد الآخرين حتى يكون هؤلاء الأفراد عناصر فعّالة في نجاح المنظومة.

وهذا النجاح يتطلب أولا حسن الاختيار، وصفة الذكاء بالطبع من العناصر الأساسية التي يجب مراعاتها.

لهذا نجد أن مديري المشاريع عند التوظيف لأي مشروع برمجي يستعينون بمجموعة من اختبارات الذكاء المختلفة وذلك لوضع الشخص المناسب في المكان المناسب.

ويوجد العديد من الاختبارات لتحديد نسبة الذكاء لدى الإنسان ومنها :

نوع الاختبار	باللغة الإنجليزية	نوع الاستعمال
اختبار الاستعداد الذهني	Aptitude Test	للتنبؤ بنجاح الشخص أو فشله في أعمال معينة.
اختبار الإنجاز	Achievement Test	لقياس نسبة استيعاب الإنجازات العلمية.

اختبار الاستيعاب الشفوي	Verbal Comprehension Tests	يختبر المقدرة على الاستيعاب الشفوي للأفراد.
اختبار الذاكرة	Memory Test	يختبر القدرة على استذكار الكلمات كأزواج أو كجمل.
اختبار الأرقام	Number Test	يختبر القدرة على التعامل بالأرقام والحسابات.
اختبار سرعة الإدراك	Perceptual Speed Test	يختبر القدرة على التعرف على التشابه والفرق بين الصور.
اختبار الاستنتاج والحجج	Reasoning Test	القدرة على الاستنتاج بناء على بعض الفرضيات.

وقبل أن يوظّف مدير المشروع فردا جديدا لمشروع ما، فمن الأفضل إجراء بعض أو كل الاختبارات المذكورة أعلاه حتى يكون التوظيف عاملًا مهمًا في نجاح المنظومة.

10.5 العمل ضمن مجموعة Working in a Group

تتم أغلب الأنشطة في هندسة البرمجيات بواسطة مجموعة من الأفراد (فريق عمل). لذلك يجب على مدير المشروع أن يقوم بتشكيل فريق ذي توازن من حيث المهارة الفنية Technical skills والخبرة Experience والشخصية Personality .

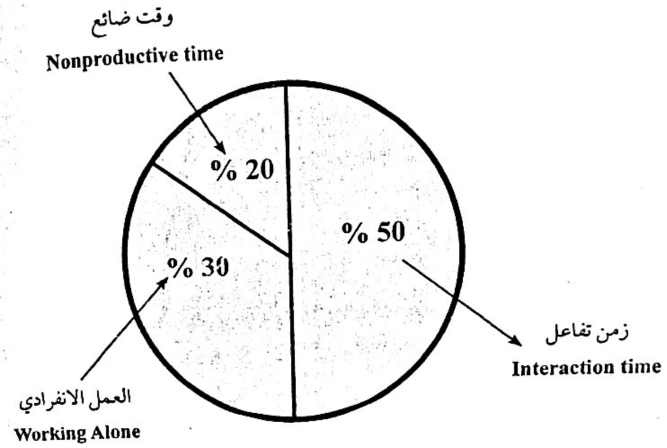
وعلى مديري المشاريع البرمجية أيضا أن يعرفوا نسبة الوقت الإنتاجي (كم ساعة يوميا مثلا) الذي يمكن أن يتوقعوه من كل فرد في فريق العمل.

يبين الشكل (10.5.1) كيف يتوزع وقت العمل بين نشاطات الفريق.

وفي هذا الشكل سنلاحظ أن نصف الوقت تقريبا يصرف في نشاطات تفاعلية بين الأعضاء مثل المناقشات والاجتماعات والتعاون.

ولكن نلاحظ أيضا أن 20 ٪ من الوقت يضيع في نشاطات غير إنتاجية مثل المحادثات الاجتماعية والمواصلات والتدريب. بينما 30 ٪ فقط للانتاج الفردي.

وعند تقدير زمن تنفيذ المشروع يجب أن نأخذ في الاعتبار أن جزءا من الوقت يتم استهلاكه في أعمال غير إنتاجية.



الشكل (10.5.1) توزيع وقت العمل ضمن مجموعة

10.6 تحفيز الشخصية Personality Motivating

ينقسم الأشخاص من حيث تحفيزهم على العمل إلى ثلاثة أنواع:

1. الشخص المحب للعمل Task-oriented person حافزه هو نجاح العمل.
2. الشخص المحب لنفسه Self-oriented person حافزه هو النجاح الشخصي.
3. الشخص المحب للتفاعل Interaction-oriented person حافزه هو وجوده مع الآخرين والتواصل معهم.

وعادة ما تكون الشخصيات الثلاثة السابقة ذات خصائص، وهي كما يلي:

1. الشخص المحب للعمل يكون مكتفيا ذاتيا ومسيطرا ومستقلا ومحبا للتنافس.
2. الشخص المحب لنفسه (للنجاح الشخصي)عادة ما يكون مجادلا وعدوانيا ومحبا للتنافس وغيور ويفضل العمل منفردا.
3. الشخص المحب للتواصل ليس عدوانيا ولكنه ذو طموح أقل في النجاح، ويحب مساعدة الآخرين.

إرشادات حول تشكيل فرق العمل لزيادة الإنتاجية:

- عادة ما يكون الرجال من نوع المحب للعمل بينما النساء من نوع المحب للتواصل.
- معظم مهندسي البرمجيات من نوع المحب للعمل.
- إن أغلب المشاريع الناجحة تتكون من أفراد من كل نوع من الأنواع الثلاثة، وتكون برئاسة شخصية محبة للعمل.
- المجموعات ذات العدد الأكبر تكون أقل نجاحا من المجموعات الصغيرة وذلك بسبب قلة التفاعل.

10.7 الشخصية غير الأنانية Egoless Personality

هناك أخلاقيات في هندسة البرمجيات لابد منها لنجاح أي منتوج برمجي.

فمثلا يجب أن يعتبر عضو الفريق في المشروع أن أي منتوج برمجي (بما في ذلك مواصفات البرامج والتصميم وتوثيق المستخدم) ملكية عامة ومسؤولية المجموعة وهو لا يعترض على التفتيش inspection من قبل أفراد مستقلين أو على التصفح من قبل زملائه walk-through.

و يجب تقبل النقد بروح عالية والعمل ضمن المجموعة وبدون روح أنانية.

وعلى سبيل المثال فإن البرمجة غير الأنانية Egoless Programming يجب أن يتم تشجيعها خاصة أثناء تفتيش ومراجعة الشفرة Code Inspection.

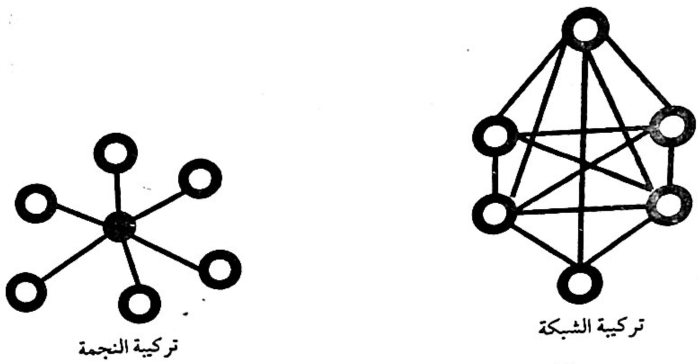
10.8 التواصل بين أفراد المجموعة Group Communication

العوامل التي تؤثر على التواصل داخل المجموعة الواحدة تشمل الآتي:

- حجم المجموعة
- تركيبة المجموعة
- شخصيات أعضاء المجموعة

ويمكن تشكيل المجموعة بإحدي الطرق التالية:

- تركيبة النجمة Star structure
- تركيبة الشبكة Network structure



الشكل (10.8.1) تركيبة الشبكة وتركيبة النجمة

في تركيبة النجمة يجب أن تمر جميع النشاطات برئيس الفريق.

ولكن في تركيبة الشبكة يقوم كل عضو بأداء مهمته منفردا مع إمكانية التفاعل مع أي عضو آخر في الفريق.

فتركيبة النجمة هي تركيبة مركزية تصلح للمجموعات الصغيرة، ولكن من الأفضل أن نستعمل تركيبة الشبكة للمجموعات الكبيرة . وتوجد أيضا التركيبة الهرمية وهي عبارة عن مدير المشروع وتحت إدارته رئيس التحليل ورئيس البرمجة. وكل من رئيس البرمجة والتحليل يتبعها مجموعة من المحللين والمبرمجين في شكل هرمي.

10.9 بيئة العمل Work Environment

تتكون بيئة العمل Work Environmentمن مكان العمل Work Place وظروف العمل Work Conditions .

ولمكان العمل وظروفه تأثير على الإنتاجية يجب عدم إهمالها من قبل مدير المشروع.

فعليه اتباع الإرشادات الآتية :

أولًا: يجب أن يتم اختيار مكان العمل بعناية فائقة لتأثيره المباشر على إنتاجية فريق إعداد المنظومة البرمجية.

أمثلة على بيئة العمل:

1. سعة الحجرة
2. الأثاث
3. درجة الخصوصية
4. درجة حرارة أو رطوبة الحجرة
5. جودة وقوة إضاءة الحجرة
6. المكيف.
7. أجهزة الاتصالات (الهاتف والفاكس والانترنت وغيرها)

ثانيًا: يجب أن يعمل أعضاء فريق إعداد البرمجيات في قاعة كبيرة بمكاتب مفتوحة، ولا يستخدم المكتب الانفرادي إلاّ لمدير المشروع (أو رؤساء فرق العمل).

انظر الشكل التالي:



الشكل (10.9.1) مكان عمل فريق إعداد البرمجيات

وهذه الوضعية كما هو واضح لها تأثير جيد على الإنتاجية والجودة لأنها تساعد على التفاعل بين أعضاء الفريق.

10.10 تصميم واجهة المستخدم: User Interface Design

يتعلق هذا النوع من التصميم بالتعامل والتواصل بين المنظومة ومستخدميها.

فقبل تصميم واجهة المستخدم والمتعلقة بتصميم الشاشات والتقارير يجب على محلل النظم ان يدرس إمكانيات ورغبات المستخدم عند التعامل مع المنظومة.

ويتعلق تصميم الواجهة بتصميم المدخلات والمخرجات على الشاشة والتقارير على الورق.

ويجب أخذ الآتي في الاعتبار:

1. التركيز على تلبية الأهداف الأساسية المتوخاة من المنظومة المراد إعدادها.
2. جعل هذه الواجهة سهلة التعلم والفهم والعمل بهاUser Friendly .
3. جعل الواجهة تؤدي جميع الوظائف المنوطة بها.

4. تستخدم الواجهة القوائم والرسومات (الأيقونات) قدر المستطاع.
5. جعل الواجهة تتسم بالوضوح وعدم الالتباس في المصطلحات.
6. توفير المساعدة Help في حالة عدم القدرة على أداء أي وظيفة أو في حالة حصول ارتباك في التعامل مع المنظومة..
7. جعل الواجهة تتعامل مع الأخطاء وتقلل حدوثها.
8. استخدام العروض التجريبية Prototypes مع الزبون لأخذ ملاحظاته وتنفيذ الصالح منها.
9. التقليل من مشاكل إدخال البيانات.
10. استخدام مصطلحات بيئة عمل الزبون.
11. تصميم شاشات جذابة للمستخدم.
12. تقليل حجم البيانات المدخلة للمنظومة.
13. إدخال البيانات الضرورية فقط.
14. استخدام التشفير بقدر الإمكان فمثلا يمكن إدخال أو اختيار (M بدلا من Male).
15. التقليل من الإدخال لأن العنصر البشرى المتمثل في المستخدم عرضة أكثر للأخطاء والحل هو عملية عرض قوائم يتم اختيار البنود المرغوبة باستخدام الفأرة.

طرق التفاعل بين المستخدم والحاسوب:

التفاعل بين البشر والحاسوب باستخدام الشاشات يمكن أن يتم بإحدى الطرق التالية :

- النص Text
- القائمة Menu
- الأيقونات Icons

كيف تصمم واجهة مستخدم رسومية جيدة؟

معظم التطبيقات البرمجية (أو المنظومات) الحديثة تتفاعل مع المستخدم عن طريق ما يعرف باسم GUI (واجهة المستخدم الرسومية)وهي نوع من أنواع التعامل مع المستخدم.

وباستخدام GUI يتم التواصل بين الناس والحاسوب عن طريق تمثيل الكائنات رسوميا (الأيقونات) واستخدام صور الأزرار والنوافذ والقوائم الأفقية والعمودية التي تظهر على شاشة جهاز الحاسوب.

واستخدام المؤشرات (مثل الفأرة) يصبح ضروريا عند استعمال GUI.

لقد أصبحت التطبيقات ذات الوسائط المتعددة شيئا فشيئا شائعة الاستعمال . إن طريقة GUIوباستخدام الرسوم، والفيديو، والصوت، والصور المتحركة (كما في نظام ويندوز) بدأت تحل محل الطرق القديمة ذات الأوامر النصية (كما في نظام دوس).

مزايا واجهة المستخدم الرسومية:

1. سهلة التعلم والاستخدام بعد دورة قصيرة.
2. يمكن أن تكون للمستخدم نوافذ متعددة على شاشة واحدة حتى يستطيع أن يتنقل من نافذة إلى أخرى.
3. إن إضافة الصورة والألوان تجعل من المنظومات أكثر جاذبية ومتعة عند الاستخدام.

إرشادات هامة عند تصميم واجهة المستخدم :

- استعمل خصائص متنوعة من GUI مثل الأيقونات والقوائم والأزرار وصناديق الاختيار وشريط السحب وشريط التقدم.
- يجب تمكين المستخدمين من التراجع عن الأمر أو إعادته.
- وفر المساعدة الفورية (خط مباشر) ولكن قلل من الحاجة إليها.
- استخدم الصوت والحركة والفيديو والنص المترابط كلما أمكن ذلك.

- خذ في الاعتبار نوع المستخدم للمنظومة . مثلا إذا كان المستخدم غير ملم بالحاسوب فيجب مراعاة قدراته ونفسيته.

مثال: منظومة مخازن

نفترض أن المطلوب إعداد منظومة المخزون تقوم بالوظائف التالية:

المعاملات Transactions

1. إضافة صنف Add item
2. تعديل صنف Edit item
3. إلغاء صنف Delete item
4. استفسار Query
5. تقارير Reports
   - تقرير عن أصناف Item report
   - تقرير عام general report

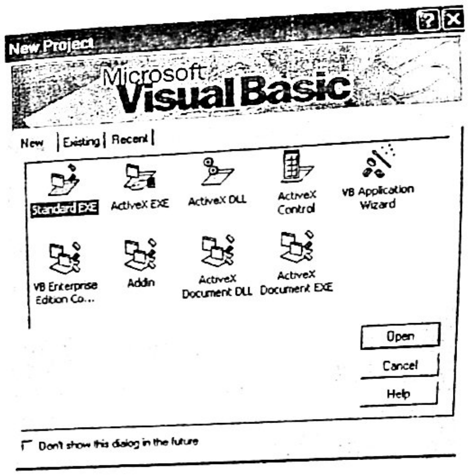
نلاحظ هنا أن المنظومة تملك وظيفتين رئيسيتين هما: المعاملات والتقارير.

في هذا المثال نستخدم بيسك المرئي 6.0 في تصميم واجهة المستخدم الرسومية.

وفي الواقع يمكننا استخدام أي لغة برمجة مرئية لهذا الغرض.

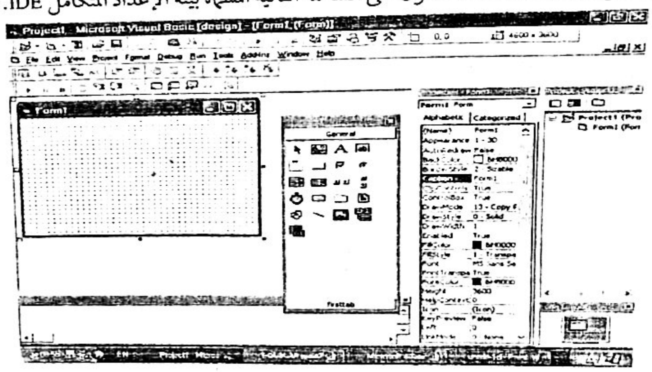
وفيما يلي الخطوات التي يمكن أن نحصل على المطلوب باستخدام بيسك المرئي.

1. من قائمة البرنامج انتق برنامج VB لتحصل على الشاشة التالية:



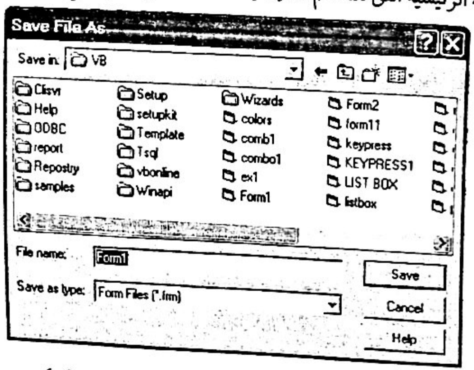
الشكل (10.10.1) الشاشة الأولى في بيئة VB

2. انتق Standard.exe للحصول على الشاشة التالية المسماة بيئة الإعداد المتكامل IDE.



الشكل(10.10.2) بيئة الإعداد المتكامل

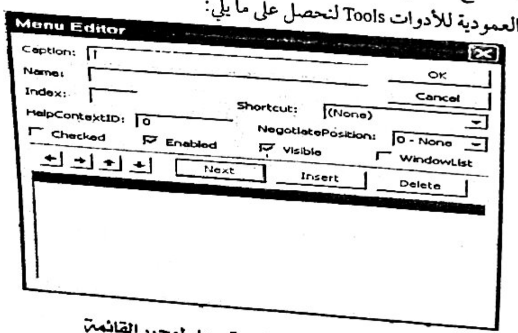
من القائمة الرئيسية انتق File ثم New project للحصول على مربع الحوار التالي:



الشكل (10.10.3) صندوق حوار (حفظ نموذج)

المطلوب هنا تخزين اسم المشروع والنموذج. اكتب example1 بدلا من form1 وبدلا من project1.

الآن نستطيع تصميم القائمة المطلوبة باستخدام menu editor محرر القائمة التي ننتقيها من القائمة العمودية للأدوات Tools لنحصل على ما يلي:



الشكل (10.10.4) صندوق حوار لمحرر القائمة

هنا يجب أن نكتب عنوانا في Caption واسما Name لأول بند في القائمة ويمكنك كتابة (معاملات) بالنسبة للعنوان وTrans للاسم ثم اضغط على زر (next) لإضافة عنصر آخر للقائمة الرئيسية وهو (تقارير) في هذه الحالة.

عندما ننفذ المشروع نتحصل على قائمة أفقية تحتوي على:

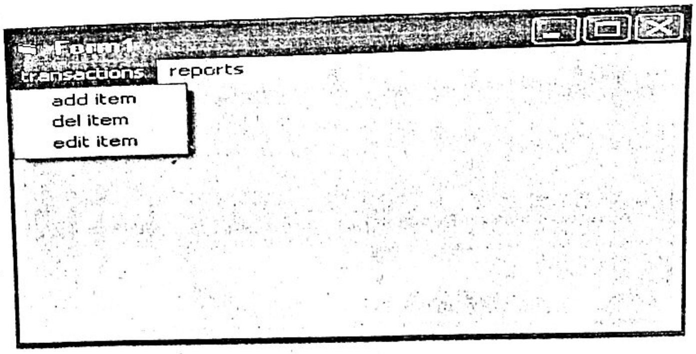
Transactions معاملات Reports تقارير

ولكن هذا لايكفي. نحتاج إلى قائمة عمودية تظهر عندما يتم النقر على المعاملات أو التقارير. لأداء ذلك، افتح محرر القائمة وأضف العنوان (...أضف صنف) بين (المعاملات) و(التقارير) مستخدما زر insert. علما بأن النقاط الواردة قبل أضف صنف تبين أنه عبارة عن عنصر متفرع من (المعاملات).

وللحصول على هذه النقاط استعمل الزر ← الموجود في محرر القائمة.

ويمكن الحصول على العناصر العمودية الأخرى للقائمة بنفس الطريقة.

والآن إذا نفذنا (المثال 1) سيعرض علينا نموذجا بقائمة أفقية تحتوي على (المعاملات) و(التقارير).



الشكل (10.10.5) قائمة أفقية وعمودية

3. نريد الآن أن نبين كيف نستخدم النموذج لإدخال البيانات. دعنا ندرس ماذا يجب أن نفعل عند النقر على (أضف صنف).

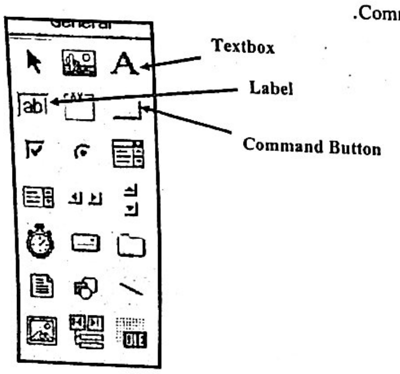
افترض أننا نريد إدخال البيانات التالية عند إضافة صنف:

- رقم الصنف
- وصف الصنف
- سعر الصنف
- كمية الصنف

وبعد إدخال البيانات نعطي المستخدم الاختيار بين حفظ البيانات أو تعديلها أو الخروج.

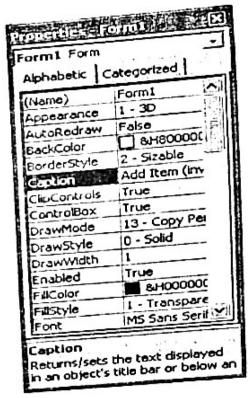
لعمل ذلك في بيسك المرئي نصمم نموذجا باستخدام أدوات التحكم Label وText Box الموجودة بصندوق الأدوات.

ولتمكين المستخدم (في هذا النموذج) من حفظ البيانات أو تعديلها أو الخروج نستعمل زر الأمر Command button.



الشكل (10.10.6) صندوق الأدوات

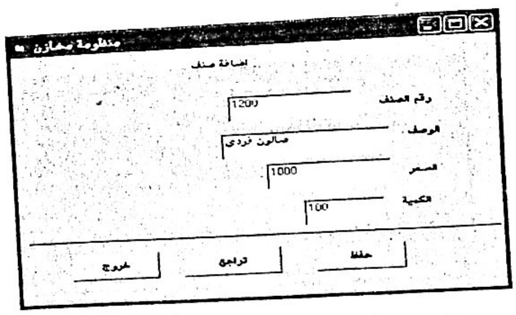
لكتابة العناوين (مثل أضف صنف Add item) استخدم الخاصية caption من شاشة الخصائص.انظر الشكل (10.10.7)



الشكل (10.10.7) نافذة الخصائص

وبنفس الطريقة يمكن كتابة العنوان المناسب في النموذج باستخدام نافذة الخصائص.

والآن عند تنفيذ المشروع والنقر على (أضف صنف) يجب أن نحصل على النموذج المبين بالشكل (10.10.8)



الشكل (10.10.8) نموذج إدخال بيانات صنف

أنواع التقارير:

1. تقارير تفصيلية Detailed reports
2. تقارير مختصرة Summary reports وهي تخص المدراء للمساعدة في اتخاذ القرار
3. كشوفات أو قوائم عند الطلب تتعلق بجزء من البيانات

10.11 أسئلة للمراجعة:

1. لماذا من المهم أن يدرس مهندس البرمجيات علم النفس؟
2. لماذا يجب على مهندس البرمجيات دراسة العوامل البشرية الخاصة بعلم النفس؟
3. ماهي العوامل البشرية الأساسية التي تؤثر على الإنتاجية؟
4. قارن بين الشخصية الموجهة نحو المهمة task-oriented والشخصية الموجهة نحو المصلحة الشخصية self-oriented.
5. ما هي النقاط التي يجب أخذها في الاعتبار عند تشكيل فريق عمل؟
6. اذكر خمسة عوامل لبيئة العمل work-environment التي تساهم في رفع الإنتاجية؟
7. ضع علامة صح وعلامة خطأ:
   - علم النفس هو علم سلوك الإنسان ( )
   - يدرس علم النفس علاقة الإنسان بالآخرين ( )
   - سلوك الإنسان ليس له علاقة بالإنتاجية ( )
8. أكمل الفراغات الآتية:
   - يختلف الأفراد في المعرفة وفي------------ وفي ------------
   - يجب على مدير المشروع أن يكوّن فريق متوازن من حيث ------------ و------------ و------------

- توزيع وقت العمل ضمن مجموعة يكون كالتالي:

٪---------- زمن تفاعل

٪---------- عمل انفرادي

٪---------- وقت ضائع

- الشخص المحب لنفسه عادة ما يكون -------، -------، -------

- أنواع الأشخاص من حيث تحفزهم للعمل:

----

9. ضع علامة صح أو خطأ:

- عادة ما يكون الرجال من النوع الغير محب للعمل ( )
- معظم مهندسي البرمجيات من النوع المحب للتواصل ( )
- المجموعة ذات العدد الأصغر تكون أقل نجاحا من المجموعة ذات العدد الأكبر ( )

10. اسرد أمثلة على بيئة العمل

11. صمم مكان يتواجد فيه فريق عمل المنظومة

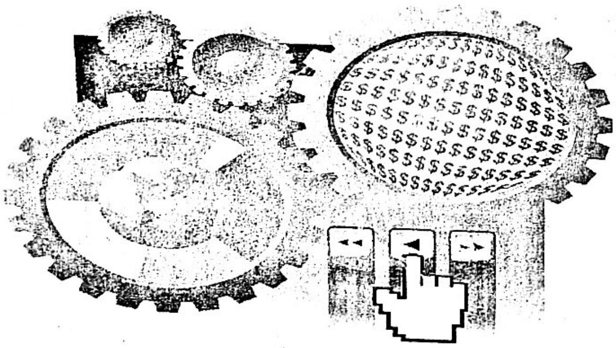
12. أذكر 5 سمات شخصية تعتقد ضرورة توفرهما في أي فرد يتم إدماجه في الفريق الخاص بالمنظومة .

13. أذكر 4 أنواع من اختبارات الذكاء عن توظيف الأفراد.

الفصل الحادي عشر

لغة النمذجة الموحدة وأدوات كيس

UML and CASE TOOLS



11.1 مقدمة

من المهم أن نلفت النظر في هذا الكتاب أن استخدام التقنية الشيئية -Technology Object Oriented قد بدأ يهيمن على مجال هندسة البرمجيات طبقا لما توقعه بيل غاتسBill Gates صاحب أكبر شركة برمجيات والمعروفة باسم ميكروسوفت MicroSoft.

وهو أسلوب جديد في التفكير عند إعداد المنظومات سواء في المنهجية أو الأدوات باستخدام العرض التجريبي Prototyping كوجه مهم في التصميم.

ومن المزايا المهمة لهذه التقنية هو إعادة استخدام الكائنات reusable objects في منظومات مستقبلية تضمن الجودة وسرعة الإنجاز.

وشمل هذا التطور العمل على تسهيل عملية إعداد البرمجيات، وذلك بإضافة أدوات تساعد المحلل والمصمم والمبرمج في عملية إعداد المنظومات تسمى أدوات كيس CASE Tools سيتم التعرض لها في هذا الفصل.

11.2 منهجيات إعداد البرمجيات Software Development Methodologies

هناك العديد من المنهجيات المستخدمة الآن في مراحل اعداد المنظومة.

وكل منهجية يتم فيها استخدام أدوات في التحليل والتصميم والبرمجة ومنها:

1. التحليل الهيكلي Structured Analysis: هو عملية تهدف لدراسة النظام بغية معرفة طريقة أدائه لعمله والمشاكل التي تعترضه وتحديد المتطلبات وذلك باستخدام منهجية التفكيك والتجزئة الهيكلية(الهرمية) باستخدام أدوات التحليل الهيكلية للنظام الحالي والمقترح .
2. التصميم الهيكلي Structured Design: هو عملية تحويل أو ترجمة المتطلبات إلى تمثيل

للمنظومة الجديدة (الحل على ورق) أي الوصول إلى نموذج معنوي يبين وظائف المنظومة ويصف الحل للمسألة المراد تنفيذها باستخدام أدوات وطرق هيكلية (هرمية).

3. البرمجة الهيكلية Structured Programming: عملية مبنية على استعمال تصميم (من فوق إلى تحت)، والبنيات التتابعية، وبنيات الاختيار(مثل: جملة إذا وإلا)، وبنية الحلقة (مثل حلقة طالما) في كتابة البرامج.

4. التحليل الشيئي Object–Oriented Analysis: هو طريقة للتعامل مع المتطلبات من وجهة نظر الفصائل classes والكائناتObjects .

5. التصميم الشيئي Object–Oriented Design: طريقة تؤدي إلى تفكيك المنظومة بمنحى كائني وذلك باستخدام الكائنات والفصائل والتفاعلات بينها .

6. البرمجة (الشيئية) (OO) Object–Oriented Programming: طريقة للبرمجة يتم فيها تنظيم البرامج كمجموعة من الكائنات (الأشياء) بحيث يمثل كل كائن حدثا لفصيلة معينة وتتم العلاقة بين الفصائل عن طريق علاقات الوراثةInhertiance .

إن مجال المنهجيات الشيئية OO methodologies لا زال لم ينضج بعد في كثير من بلدان العالم، فلا زالت المنهجية الهيكلية هي السائدة وكل فريق يستخدم الأدوات الخاصة بمنهجيته.

وشركة ميكروسوفت تستخدم منهجية خاصة بها أعدتها لاستخدامها وتسويقها تسمى Microsoft Foundation Classes (MFC) وهي عبارة عن مكتبة فصائل class library لإعداد المنظومات.

أي منهجية يجب أن نستعمل؟

الإجابة سهلة: انتق المنهجية التي تناسب تطبيقك، وابدأ في استخدام هذه المنهجية بعد التدرب عليها جيدا .

ملاحظات هامة:

■ إن استخدام التقنية الشيئية في التحليل والتصميم والبرمجة لا يتناسب مع النموذج

التدفقي التسلسلي Waterfall model في إعداد المنظومات بل إن نموذج العرض التجريبي Prototyping model هو الأنسب لذلك .

■ للتغيير من منهجية (مثل المنهجية الهيكلية السائدة) إلى منهجية أخرى مثل المنهجية الشيئية يجب دفع ثمن ذلك وهو: تدريب الناس على هذه التقنية الجديدة وتجميع أو شراء عدد من المكونات البرمجية الجاهزة ووضعها في المكتبة الالكترونية Component libraries ليتم استخدامها فيما بعد.

■ ليس هناك فاصل حقيقي بين التحليل والتصميم عند استخدام المنهجية الشيئية في إعداد المنظومات. فمثلا خلال التحليل يتم التعرف على الكائنات الخاصة بالنظام وخصائصها والعلاقات بين هذه الكائنات حيث يتم إضافة تفاصيل لها في التصميم .

■ بعض العلماء في هندسة البرمجيات أجاز مزج أدوات من منهجيات مختلفة . فمثلا يمكن استخدام مخطط انسياب البيانات من المنهجية الهيكلية مع مخطط استخدام الحالة من المنهجية الشيئية.

والمؤلف من رواد المنهجية الجديدة وسماها البعض المنهجية الهجينة Hybird Methodology.

11.3 التقنية الشيئية OO Technology

تعتبر التقنية الشيئية OO Technology نهجا في التعامل مع إعداد البرمجيات لتقريب صورة النظام المراد برمجته من الواقع العملي في الحياة اليومية، حيث نعرف أن هذا الكون يتكون من كائنات ملموسة، وغير ملموسة، وكل كائن له خصائصه التي تصف حالته وعمليات يجريها أو تجرى عليه.

وهدف هذه التقنية الشيئية (الكائنية) OO technology تحقيق الأهداف التالية:

1. برمجيات زهيدة التكلفة Low-cost software
2. سرعة إنجاز البرمجيات Fast development
3. برمجيات أكثر اعتمادية More reliable software

4. برمجيات أعلى جودة Higher quality software
5. سهولة تغيير البرمجيات Changeable software
6. بساطة تصميم النظم Ease of design
7. برمجيات ذات مرونة Flexible software

وقد تطورت هذه التقنية إلى ما يعرف الآن بلغة النمذجة الموحدة UML (Unified Modeling Language) وهي عبارة عن مجموعة مخططات Diagrams ومواصفات Descriptions في شكل سيناريوهات scenarios قد تمّ الاتفاق عليها من قبل علماء هندسة البرمجيات كأدوات قياسية standard tools تستخدم في تحليل وتصميم وتنفيذ النظم الشيئية من قبل أشهر علماء هندسة البرمجيات المتخصصين في النظم الشيئية.

ويتم الآن تجهيز العديد من الأنظمة (التطبيقات البرمجية) ذات الدرجة العالية من التعقيد والتعددية والجودة باستخدام أدوات UML.

ومن أمثلة هذه الأنظمة:

1. أنظمة المحاسبة والجباية والمالية
2. ألعاب الفيديو
3. أنظمة الحجز المختلفة
4. الأنظمة المصرفية
5. الذكاء الاصطناعي
6. التعليم الالكتروني
7. التجارة الالكترونية
8. البريد الالكتروني
9. الإدارة الالكترونية

10. أنظمة التشغيل

11. الرسوم المتحركة

مع ملاحظة أن النظم الموزعة Distributed systems يتم فيها استخدام الشبكات والانترنت.

وهذه الظاهرة فرضت استعمال لغات برمجة حديثة عادة ما تكون مرئية visual وشيئية OO خصوصا بعد ازدياد عددها وتنوع المتطلبات في أواخر الثمانينات والتسعينيات.

ملاحظة: من خلال خبرتي كمحلل نظم أود أن أعلق على هامش هذا البند بأن الزبون الآن هو غير الزبون الذي عاصرناه في السبعينيات. فالآن نجد أن له خلفية واسعة بالحاسوب والتقنية بوجه عام، ويكفي أنه على اطّلاع واسع بإمكانيات الحاسوب، لهذا فلا غرابة في تنوع وتعقد المتطلبات التي يريدها.

ويبين الجدول (11.3.1) أهم المساهمات في إعداد البرمجيات في نهاية القرن العشرين.

المساهمة	السنة
لغات برمجية شيئية: Smalltalk Stimula	1970
أكثر من أداة تحليل وتصميم وبرمجة شيئية قد تمّ إعدادها	1980
ظهور أشهر منهجيات التحليل والتصميم الشيئي Booch، Rum Baugh and Jacobson	1992
ظهور UML (Unified Modeling Language) بواسطة العالمين: Rumbaugh and Booch	1995
توحيد المنهجية في التحليل والتصميم الشيئي UML as OOAD standard	1997

الشكل (11.3.1) أهم المساهمات في إعداد البرمجيات

نستطيع أن نلاحظ من الجدول في الشكل (1.3.11) أننا نعيش في عصر التقنية الكائنية object-oriented technology.

وهذه التقنية تغطي معظم عناصر إعداد البرمجيات، وبالتحديد التحليل والتصميم ولغات البرمجة وقواعد البيانات والشبكات.

وبعبارة أخرى، فإن النمذجة الهيكلية Structured modeling يجري استبدالها بالنمذجة الشيئية Object-orieted modeling، كلما أصبحت المشاريع البرمجية أكثر تعقيدا وتعددية، وكان الحرص على الجودة وسرعة الإنجاز من أهم متطلباتها.

وتوجد العديد من المصطلحات المستخدمة في التقنية الشيئية نورد منها مايلي:

1. الفاعل أو الممثل Actor: شخص ما (أو نظام آخر) خارج النظام ويتفاعل معه بأدائه لوظائف معينة خاصة بالنظام.
2. المعلومة Artifact: جزء معلوماتي يستخدم في (أو ينتج عن) عملية إعداد برمجيات.
3. الفصيلة Class: تجريد وحيد ومعرف يصف فئة من الأشياء لها نفس الخصائص attributes والعمليات operations والعلاقات relationships والقواعد اللغوية semantic.
4. الكائن Object: رزمة برمجية تم تصميمها وتحتوي على البيانات والعمليات(الخدمات) المنوطة بها وهي حالة خاصة للفصيلة.
5. النمذجة Modeling: تمثيل أو تصميم التطبيقات البرمجية قبل كتابة شفرتها.
6. لغة النمذجة الموحدة UML هي لغة تستخدم أدوات رسومية ونصية شيئية لتوثيق وتحليل وتصميم وتنفيذ معلومات المنظومة.
7. الرسائل Messages: وهي آلية تواصل بين كائنات المنظومة الشيئية.
8. الطرق Methods عبارة عن خدمات يؤديها الكائن لتلبية المتطلبات الوظيفية.
9. الوراثة Inheritance مفهوم في التقنية الشيئية حيث ترث الفصيلة الفرعية البيانات والطرق

من الفصيلة الفوقية (الأم). ومن مزايا هذا المفهوم أن الفصيلة الفرعية (الكائن) تستطيع إعادة استعمال سلوك الفصيلة الأم.

10. التغليف Encapsulation شكل من أشكال إخفاء المعلومات بين الفصائل (الكائنات) ويتم فقط تبادل البيانات عن طريق الرسائل messages بين الكائنات المختلفة مع إخفاء البيانات الداخلية الخاصة بالكائن عن الكائنات الأخرى.
11. تعدد الأشكال Polymorphism:

هذه الكلمة في حد ذاتها مصطلح إغريقي قديم ويعني تعدد الأشكال. أما استخدامه في التقنية الشيئية فيعني أن هذه الخاصية تسمح بأداء عدة وظائف (أو عمليات) تحت مسمى واحد.

فمثلا قد يكون المسمى «إيجاد مساحة»: فباستخدام مفهوم تعدد الأشكال يمكن لنا استخدام مكون برمجي كائني واحد فقط في إيجاد مساحة الدائرة ومساحة المثلث ومساحة المستطيل.

11.4 مخططات UML ماهي مصطلح UML

لقد بيّنا في البند السابق أن أدوات UML عبارة عن مجموعة مخططات تستخدم في التحليل والتصميم للنظم الشيئية.

ومن أمثلة هذه الأدوات: مخطط استخدام الحالة UCD ومخطط الفصيلة Class Diagram، ونستعرض هنا بعضا من مخططات لغة النمذجة الموحدة Unified Modeling Language وهي تربو على عشر مخططات نذكر منها مايلي:

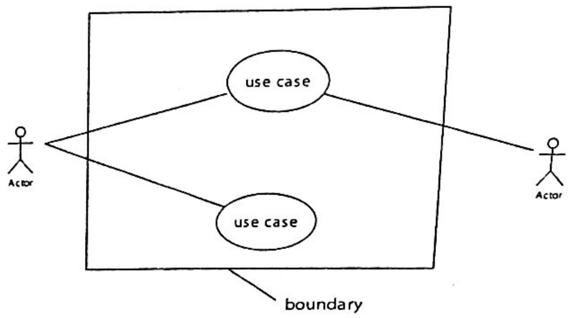
- **مخطط حالة استخدام Use Case Diagram** أو باختصار UCD: هو أداة تصف المتطلبات الوظيفية للنظام.

وبتعريف آخر هو أداة تحليل كائنية تستخدم لغرض تحديد المتطلبات الوظيفية للنظام ونطاق المشروع.

ويستخدم تقريبا في كل مشروع خاصة في التحليل كوسيلة اتصال بين المستخدم ومحلل النظم عند تعريف وتحديد المتطلبات الوظيفية.

ويتكون UCD من حالات استخدام (وظائف) وممثلين (مستخدمين أو نظم خارجية) في تفاعل مع بعضهم البعض لإنجاز معاملات النظام.

الشكل العام لمخطط حالة الاستخدام كما يلي:



الشكل (11.4.1) مخطط حالة استخدام

رموز مخطط حالة الاستخدام:

الممثل actor

الممثل هو دور يلعبه المستخدم أو نظام خارجي مع النظام المقترَح.

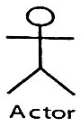
مثال:

الممثل يمكن أن يلعب أحد الأدوار التالية :

- ■ موظف Employee
- ■ موظف خزينة Cashier
- ■ زبون Client

لاحظ أن:

- ■ الممثل الواحد يمكن أن يمثل عدة مستخدمين أو أنظمة خارجية.
- ■ المستخدم الواحد أو النظام الخارجي الواحد يمكن أن يلعب أدوارا متعددة.



حالة استخدام Use Case:

هي عبارة عن إجراء عن طريقه يتفاعل الممثلون مع النظام. وهو بلغة تحليل النظم الهيكلية عبارة عن معاملة أو وظيفة.

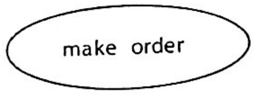
والشكل الذي يمثل حالة استخدام يكون بيضاويا على النحو التالي:



الإجراء الذي يتم تنفيذه في هذه الحالة يكتب داخل الشكل البيضاوي كما يلي:



- ■ افحص الصنف Check item
- ■ ادخل تفاصيل الطلبية Enter order details
- ■ عمل طلبية Make order



حالة الاستخدام (عمل طلبية) يمكن أن تكون ملخصا للخطوات التالية :

1. انتق صنفا Select item
2. اتصل برجل المبيعات Call salesman
3. جهز الفاتورة Prepare invoice
4. ادفع الفاتورة Pay invoice

نرى هنا أن هذه الخطوات مرتبة تسلسليا:

مثال:

معاملة (حالة استخدام) سحب صك في منظومة مصرف

حالة الاستخدام سحب صك هي ملخص للخطوات التالية:

1. تسليم الصك لموظف الخزينة
2. التأكد من بيانات الصك
3. التأكد من رصيد الزبون
4. تعديل رصيد الزبون
5. تسليم النقود للزبون

ملاحظات هامة:

1. لكل ممثل يوجد على الأقل حالة استخدام واحدة .

2. من المهم أن نفرق بين المستخدم والممثل، المستخدم هو إنسان يقوم باستخدام النظام، أما الممثل فيمكن أن يكون مستخدما أو نظاما خارجيا يتصل بالنظام المقترح.

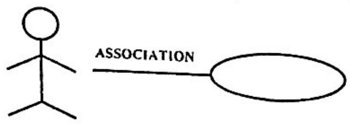
العلاقات Relationships

توجد 3 أنواع من العلاقات:

1. علاقة ممثل وحالة استخدام (تسمى علاقة ربط association).
2. علاقة حالة استخدام وأخرى (تسمى علاقة امتداد Extend أو شمول Include).
3. علاقة ممثل بآخر (تسمى علاقة تعميم generalization).

علاقة الربط Relationship Assocation

يتم تمثيل العلاقة بخط يربط بين الممثل وحالة أو حالات الاستخدام وذلك لتوضيح أن الممثل هو الذي يقوم بحالة الاستخدام.



وحالة الاستخدام التي يقوم بها ممثلون عديدون يتم توضيحها كما يلي:

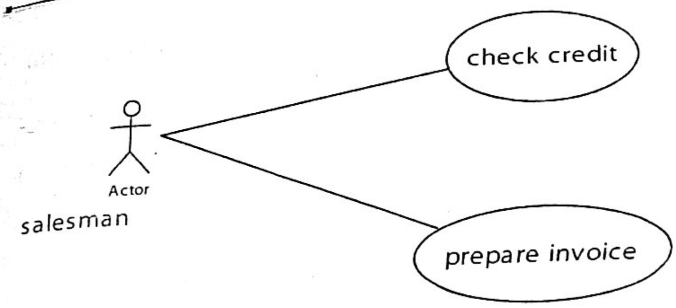
اعرض الدرجة View Grade



في الشكل التالي الممثل {البائع} يقوم بحالات استخدام متعددة :

افحص الرصيد Check credit

جهز الفاتورة Prepare invoice



يمكن أن توجد علاقة بين حالة استخدام وأخرى.

علاقة اشتمال / امتداد Extend \ Include Relationship

وهناك نوعان من العلاقة بين حالات الاستخدام وهما:

1. علاقة الشمول   <u>include</u> relationship

2. علاقة الامتداد   <u>extend</u> relationship

علاقة شمول Include Relationship

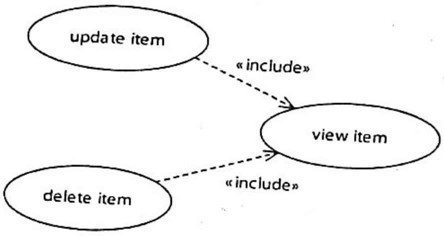
علاقة الشمول تستخدم لتبين أن حالة استخدام { تسمى إما الأساسية أو الوالد} تشمل وظائف موجودة في حالة (أو حالات) استخدام أخرى (تسمى الأبناء).

يجب ملاحظة أن في بعض الأحيان نجد كلمة use مكافئة لكلمة include.

وفي هذا الفصل سوف نستخدم الاختيار include.

مثال:

في المخطط التالي لدينا حالة استخدام (اعرض الصنف) مستخدمة من قبل حالتي استخدام وهما (احذف الصنف) و(عدّل الصنف)..



الشكل (11.4.2) علاقة الشمول

علاقة الامتداد Extend Relationship

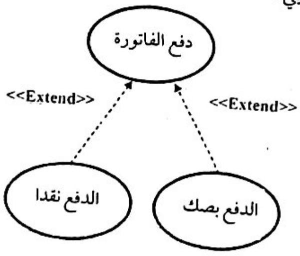
علاقة الامتداد بين حالة استخدام وأخرى تحدث عندما يكون لدينا أحد الأوضاع التالية:

- حالة استخدام رئيسية لديها حالات استخدام بديلة (خيارات).
- حالة استخدام إضافية يمكن إضافتها لحالة الاستخدام الرئيسية.

مثال :

منظومة مبيعات

حالة الاستخدام التالية (ادفع الفاتورة) في نظام مبيعات يمكن أن يكون لها حالتا استخدام بديلتان كما هو مبين فيما يلي:

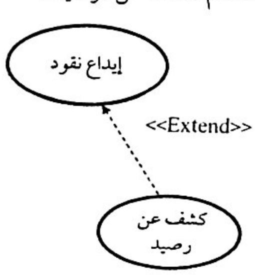


الشكل (11.4.3) علاقة امتداد

مثال:

منظومة مصرفية

حالة الاستخدام الرئيسية (إيداع النقود) في نظام مصرفي يمكن أن يكون لها حالة استخدام إضافية (امتداد) مثل حالة الاستخدام (كشف عن الرصيد).



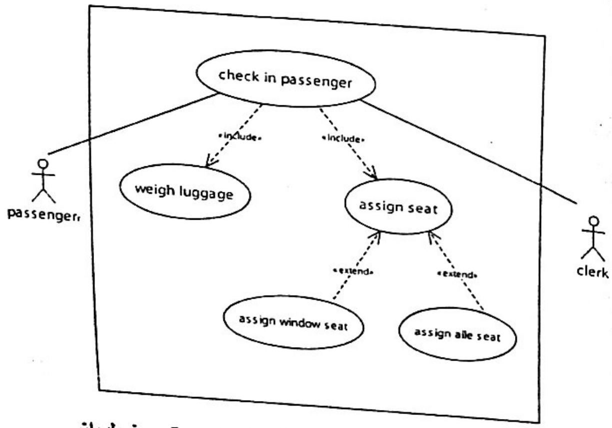
لاحظ أن حالة الاستخدام (كشف عن رصيد) تعتبر غير ضرورية في حالة (إيداع نقود) بل هي حالة اختيارية، وهذا يعني وجود علاقة امتداد بينهما.

مثال:

منظومة الخطوط الجوية

في هذا المنظومة لدينا حالات الاستخدام التالية:

إجراءات الصعود	Check-in passenger
زن الحقائب	Weigh luggage
خصص مقعد	Assign seat
خصص مقعد جانب النافذة	Assign window seat
خصص مقعد جانب الممر	Assign aisle seat
مسافر	Passenger
موظف	Clerk



الشكل (11.4.4) مخطط حالة استخدام لمنظومة حجز طيران

علاقة التعميم:

التعميم Generalization هو علاقة وراثة، وهو علاقة تمثل لمثل Actor-to-actor

مخطط الفصيلة:

مخطط الفصيلة Class diagram هو أداة تحليل وتصميم كائني لوصف الهيكلية الساكنة للنظام.

في هذا النوع من المخططات نرسم مجموعة من الفصائل ونبين علاقاتها.

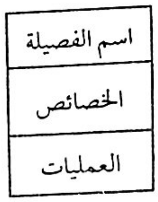
ويمكننا تصور مخطط الفصيلة على شكل شجرة العائلة التي تتكون من أفراد العائلة والعلاقات بينهم.

■ **رموز مخطط الفصيلة Class Diagram**

يتم تمثيل الفصيلة على شكل مستطيل مقسم إلى ثلاثة أقسام:

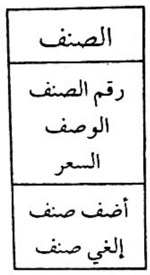
اسم الفصيلة Name of the class، الخصائص Attributes، العمليات Operations

كما يلي:



فمثلا يمكن أن تكون فصيلة الصنف كما يلي :

فصيلة الصنف:



رؤية الفصيلة:

تبين رؤية الفصيلة من له القدرة على الوصول إلى البيانات في داخل الفصيلة.

وهناك ثلاث طرق للوصول للبيانات الخصائص والطرق (في الفصيلة):

1. رؤية خاصة Private (تمثل بإشارة -):

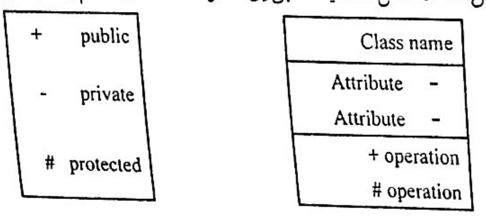
يوجد لدينا هنا فصيلة واحدة تخفي معلوماتها (الخصائص والعمليات) عن الفصائل الأخرى.

2. رؤية عامة: Public (يتم تمثيلها بإشارة +)

هنا يمكن أن ترى الفصائل الأخرى معلومات هذه الفصيلة.

3. رؤية محمية protected (يتم تمثيلها بإشارة #)

يمكن هنا أن تصل الفصيلة الابن إلى معلومات الفصيلة الأم بعملية الوراثة.



علاقات الفصائل: Class Relationships

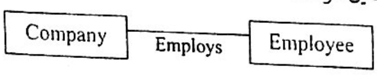
توجد أنواع مختلفة من العلاقات بين الفصيلتين:

علاقة ربط: Association

يتم تمثيل علاقة الربط Association بخط مستقيم بين الفصيلة والأخرى.

وعلى سبيل المثال، يمكن أن تكون العلاقة بين الموظف والشركة على النحو التالي:

الشركة تعين الموظف



التضاعف multiplicity (cardinality) هو عدد مرات أحداث فصيلة ذات علاقة بأحداث فصيلة أخرى.

ويمكن أن يكون التضاعف من أحد الأنواع التالية:

1 : no more than one واحد فقط

* : more than one أكثر من واحد

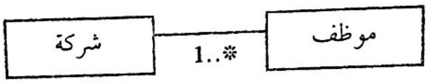
1 .. 0 : from zero to one من الصفر إلى الواحد

* .. 0 : from zero to many من الصفر إلى ما فوق

* .. 1 : from one to many من الواحد إلى ما فوق

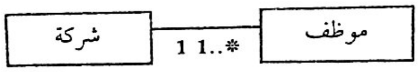
مثال على تضاعف من نوع -1*‌

الموظف يتم تعيينه من شركة واحدة ولكن الشركة يمكن أن تعين العديد من الموظفين.



لاحظ أن الشركة يمكن أن تعين أكثر من موظف واحد.

مثال على مضاعفة من نوع -1 1..*



فصيلة الربط: Association class

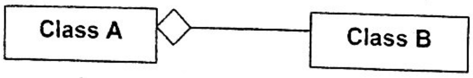
إذا افترضنا أن هناك علاقة ربط بين فصيلتي (الطالب والمقرر الدراسي) مثل:

الطالب يلتحق بمقرر دراسي، لهذا سيكون لدينا فصيلتان هما فصيلة المقرر الدراسي وفصيلة الطالب ،والرباط بين هاتين الفصيلتين هو رباط الالتحاق. وهذا الرباط له خاصيتان هما: تاريخ الالتحاق ودرجة المقرر، ويمكن أن يكون له عمليات أيضا. لهذا من الأفضل أن يتم تمثيل هذا الرباط بفصيلة تسمى فصيلة الربط...

علاقة تجميع Aggregation Relationship:

هي عملية ربط بين فصيلتين أو أكثر Aggregation Relationship بحيث تمثل فصيلة واحدة الكل، بينما يمثل الآخرين أجزاء من الكل.

الكل هنا يمثل الأب، والأجزاء تمثل الأبناء.

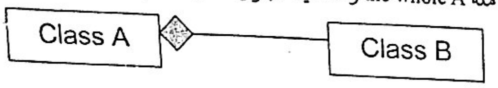


مثال: المعالج المركزي هو جزء من الحاسوب.



علاقة تركيب: Relationship Composition

يعتبر هذا النوع نوعا خاصا من علاقة التجميع aggregation مع وجود علاقة قو
الفصيلة الشاملة the whole A والفصيلة الجزئية its part منها B.

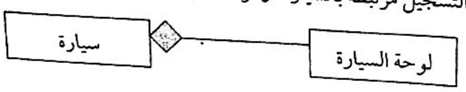


لاحظ هنا أن:

- الأجزاء تنتمي إلى فصيلة شاملة واحدة فقط.
- الأجزاء عادة ما تحيا وتموت مع الفصيلة الشاملة.

مثال: على علاقة التركيب

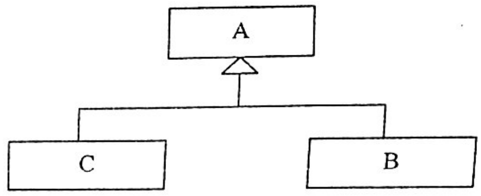
- العلاقة بين السيارة ولوحة التسجيل المعدنية.
- لوحة التسجيل مرتبطة بالسيارة، وتموت بموتها.



التعميم Generalization:

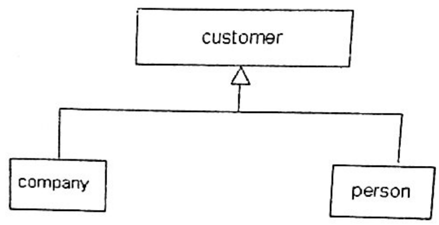
هو اسم آخر للعلاقة الوراثية.

في هذه العلاقة، الأبناء B وC يرثون الوظائف العامة المعرفة للفصيلة الأم A.



مثال على علاقة التعميم:

فصيلة الشركة وفصيلة الفرد العادي لهما علاقة تعميم مع فصيلة الزبون على النحو التالي:



- ■ المخطط التسلسلي Sequence Diagram

  وهو مخطط يصف خطوات حالة استخدام محددة موضحا الممثلين الذين يؤدون هذه الخطوات.

- ■ مخططات المكونات Component Diagrams المستخدمة في توضيح كيف تتفاعل مكونات النظام مع بعضها.

ولرسم هذه المخططات من المناسب أن نستعمل رزمة كيس Case tools التي تساعد على رسم هذه المخططات بصورة أدق بدلا من استخدام برنامج عام مثل word أو paint.

ومن حسن الحظ ونظرا لارتفاع سعر هذه الأدوات توجد طرق أخرى بديلة لاستخدام هذه التقنية وخصوصا للطالب وهي موجودة في نظام المكتب MS-OFFICE مثل البرمجية VISIO أو يمكن للطالب تحميل برنامج مثل Smartdraw أو Pacestar من الانترنت وهو يؤدي الغرض المطلوب.

وإنني أعتقد أن أسلوب UML لهندسة البرمجيات يحتاج إلى نظرة متعمقة أكثر مما تعرضت له في هذا الفصل، والمهتم بهذا الموضوع سيجد كتبا ومواقع في الانترنت تتعامل مع هذا الموضوع بتفاصيل أكثر.

والجدول التالي يتضمن بعض الأدوات المهمة التي تلزم معد المنظومة لإعداد البرمجيات الحديثة.

الأدوات التالية يجب أخذها في الاعتبار عند إعداد نظم حديثة:

البرمجة المرئية	Visual programming
أدوات كيس	CASE tools
مولدات الشفرة	Code generators
اللغات الإجرائية	Nonprocedural languages
إعادة الهندسة	Re-engineering
مكتبة الفصائل	Class library
تقنية الخادم والزبون	Client-server technology

11.5 نموذج الطريقة القويمة للمعالجة الموحدة
Rational United Process Model

يعتبر نموذج RUP Model وسيلة كائنية لإعداد البرمجيات بواسطة عملية الإعداد التزايدي التكراري وأدوات UML المستخدمة في مرحلة التحليل والتصميم والتنفيذ في البرمجة الكائنية.

وهي تتكون من المراحل التالية:

- ■ مرحلة الابتداء: **Inception phase**

  تستخدم لتعريف النظام بما في ذلك المحتويات والهدف.

  - التحليل المبدئي
  - تحديد المتطلبات (باستخدام DFD ومخططات أخرى)

■ مرحلة التفصيل: Elaboration phase

هذه هي المرحلة التي يتم فيها التصميم، والمراحل الفرعية لهذه المرحلة هي:

- قسّم النظام إلى نظم فرعية.
- ترجم المتطلبات إلى تصميم باستخدام المخطط الهيكلي أو UCD

■ مرحلة البناء Build phase :

في هذه المرحلة يتم إعداد الشفرة (كتابة البرمجيات).

■ مرحلة التسليم Delivery phase :

يتم تسليم المنظومة إلى الزبون.

11.6 أدوات كيس CASE Tools

تشبه أدوات كيس البرمجية برنامج AUTO-CAD الذي تستخدم لمساعدة المهندسين في رسم الخرائط المعمارية والإنشائية بمساعدة الحاسوب.

ولأداء أي منتوج برمجي (منظومة) نحتاج عادة إلى أدوات وكلما كانت هذه الأدوات متقدمة، كلما كان المنتوج البرمجي تحت الإنشاء أكثر اعتيادية more reliable فعلى سبيل المثال لا الحصر نجد أن المهندس المعماري العصري عندما يصمم مبنى كبيرا أو برجا يحتاج إلى برمجية Auto-CAD فهي دقيقة وتوفر الوقت والجهد.

وتوفر هندسة البرمجيات رزمة من البرمجيات مثل CAD لاستخدامها في إعداد وصيانة المنظومات أي رسم المخططات باستخدام الحاسوب في التحليل والتصميم وتصميم الشاشات والتقارير وبالتالي تزداد الكفاءة والجودة.

وتسمى هذه الأدوات (CASE). بهندسة البرمجيات بمساعدة الحاسوب Computer Aided Software Engineering أي إعداد المنظومات بمساعدة الحاسوب.

ومن أمثلة أدوات كيس التي تستخدم لرسم مخطط حالة الاستخدام ما يسمى برنامج VISIO من إعداد شركة ميكروسوفت وهو من ضمن رزمة OFFICE.

11.7 تاريخ أدوات كيس CASE

اسرد تاريخ كيس

في السبعينيات كانت بداية CASE وكانت عبارة عن معالج كلمات Word processor يستعمل لتوثيق البرمجيات (المنظومات) التي تم إعدادها في تلك الآونة. ثم تدريجيا أصبحت تشمل على الآتي:

1. برامج خدمات Utility programs مثل المترجمات ونظم التشغيل.
2. أدوات رسم مخططات Diagramming tools مثل رسم مخطط انسياب البيانات والمخطط الهيكلي.
3. المحررات editors وكاشفات الأخطاء Debuggers.
4. مكوّنات الشاشات والتقارير Screen / report generators.
5. أدوات برمجة شيئية مثل ++C.
6. مستعرضات الانترنت Browsers.
7. أدوات إدارة المشاريع Project management tools مثل Project التابع لشركة Microsoft لرسم Gannt chart مثلا.
8. قاموس البيانات Data Dictionary.
9. أدوات الاختبار Testing Tools.
10. مكوّنات الشفرة Code Generators.

11.8 مزايا أدوات كيس CASE

يعتبر استخدام أدوات CASE من الأمور المهمة عند إعداد المنظومات الراقية للمصارف وشركات الطيران وغيرها ولها العديد من الميزات التي من أهمها:

1. أنها تبسط عملية إعداد وصيانة البرمجيات.
2. أنها تسرّع عملية إعداد البرمجيات.
3. أنها تزيد من إنتاجية المبرمج ومحلل النظم.
4. أنها توفر تواصلا أفضل مع المستخدم بسبب استخدام الرسوم إلى جانب النص.
5. أنها تشجع على استعمال البيئة المتفاعلة (مثل القوائم والأيقونات).
6. أنها توفر معايير وجودة.

11-9 أسئلة للمراجعة

1. تكلم عن المنهجية الهيكلية؟
2. ما معنى مصطلح التقنية الشيئية وماهي مزاياه؟
3. ما معنى مصطلح UML ؟
4. ماذا تعني كلمة CASE ؟ اذكر مثال لهذه الأداة.
5. اسرد تاريخ CASE .
6. اذكر الأدوات الأساسية في CASE.
7. ماهي UML؟ وكيف تختلف عن الأدوات الهيكلية؟
8. هل ترى أن DFD لا زالت صالحة في عالم البرمجة الشيئية؟ وّضح إجابتك؟
9. ما وظيفة أدوات Case في إعداد وصيانة المنظومات؟
10. اذكر أهم مكوّنات Case .
11. اذكر 6 مزايا لأدوات Case.

الخاتمة

أخيرا:

لقد تناول هذا الكتاب موضوع هندسة البرمجيات بشيء من التبسيط والإيجاز، وأعتقد أن هذه التقنية لها القدرة على صنع العديد من التطبيقات البرمجية بجودة عالية وتكلفة منخفضة ووقت قصير نسبيا، باستخدام أدوات ومعايير وعمليات وفنيات ولغات برمجة وقواعد بيانات. وهي علم متحرك ومتجدد وفي نمو متزايد.

وهناك عدة أبحاث وفروع من هذا العلم لم أتمكن من سردها جميعا ولكن لابأس من عرض عناوينها ليتمكن الطالب الباحث من الولوج إلى الانترنت والتحقق منها ومتابعة تطورها ومنها:

- هندسة المعلومات Information Engineering
- هندسة البرمجيات النظيفة Clean-Room Software Engineering
- هندسة البرمجيات العكسية Reverse Software Engineering

وقد تمت بفضل الله رحلتنا مع هندسة البرمجيات. لقد كانت رحلة مضنية ولكن ممتعة في نفس الوقت، زرنا خلالها إحدى عشر موضوعا مهما في هذا المجال. ويمكن تلخيصها في النقاط التالية:

1. الجودة Quality مهمة في عالم إعداد البرمجيات، فحاول أن يكون الاختبار والفحص والمراجعة والتحقق على التوثيق وعلى جهاز الحاسوب (أي في جميع مراحل المشروع).

2. إذا عزمت على كتابة منظومة برمجية، فعليك بالصبر والتأني. خذ وقتك في التخطيط والتحليل والتصميم وكتابة الكود (الشفرة) والاختبار.

3. اتبع نموذجا مناسبًا يروق لك من نماذج هندسة البرمجيات.

4. ضع في الاعتبار أن قبول المنظومة بيد الزبون. لذلك اعتبره عضوا في فريق العمل من البداية إلى النهاية.

والله ولي التوفيق

الملاحق

APPENDICES

ملحق (1)

المصطلحات

Terminology

- A -

Abstraction	تجريد -عموم
Activity	نشاط
Algorithm	خوارزمية
Alpha testing	اختبار ألفا
Analysis	تحليل
Application software	برمجية تطبيقية
Architectural design	تصميم معماري
Access	وصول
Attribute	خاصية
Acceptance	قبول

- B -

Base–line document	وثيقة نهاية المرحلة
Black-box Test	اختبار الصندوق الأسود
Beta testing	اختبار بيتا
Bottom--up design	تصميم من أسفل إلى أعلى
Bubble chart	مخطط فقاعي
Boundary	حدود

- C -

CASE: Computer Aided Software Engineering	هندسة برمجيات بمساعدة الحاسوب
Coupling	اقتران
Class	فصيلة
Cohesion	التجانس
Component	جزء - قطعة
Configuration management	إدارة مكونات البرمجيات
Construction	بناء البرمجية
Context chart	مخطط بيئي
Coupling	الاقتران
Critical path	المسار الحرج
Constraints	قيود
Communication	تواصل - اتصال
Connection	ربط
Content Coupling	اقتران المحتوى
Control Coupling	اقتران تحكم
Code	شفرة
Customer	زبون
Client	زبون
Change	تغيير
Class Library	مكتبة الفصائل

- D -

Data	البيانات
Developer	معد المنظومة
Data dictionary	قاموس البيانات
Debugger	كاشف الأخطاء

Design	تصميم
Documentation	توثيق
DBMS: Database Management System	نظام إدارة قواعد البيانات
Decision table	جدول القرار
Decision tree	شجرة القرار
Directing	توجيه
Development	إعداد- بناء
Document	مستند
Domain	نطاق
Driver	وحدة برمجية للتحكم

- E -

Estimation	تقدير (تقدير تكلفة مثلا)
Encapsulation	تغليف-كبسلة
Enhancement	تحسين
Environment	بيئة
EntityRelationship Diagram	مخطط الكائنات العلائقية

- F -

Failure	فشل - عطل
Fault	خلل - خطأ
Feasibility study	دراسة الجدوى
Feedback	تغذية مسترجعة
Field	حقل
Feature	ميزة- سمة
Fan-in	مروحة مقفلة
Flag	راية
Functional Requirements	متطلبات وظيفية

Form	نموذج
Freeware	برمجيات مجانية
Function	وظيفة
Flexibility	مرونة

- G -

Gantt chart	مخطط غانت
Generalization	تعميم
Graphic User Interface	واجهة المستخدم الرسومية
Generation	جيل
Glass box test	اختبار الصندوق الزجاجي
Group	مجموعة

- H -

Human Factors	العوامل الإنسانية
Hardware	مكونات مادية
Hierarchy chart	المخطط الهرمي
Hierarchal Input Process Output	هرمية الإدخال والمعالجة والإخراج

- I -

implementation phase	مرحلة التنفيذ
Integration Test	اختبار التكامل
Inheritance	توريث - وراثة
Inspection	تفتيش - (مرجعة منهجية)
Interface	واجهة-بينية

- L -

Low-level Language	لغة متدنية المستوى

- M -

Maintenance phase	مرحلة الصيانة
Manual	دليل
Methodology	المنهجية
Milestone	الزمن المخصص لانتهاء نشاط
Myth	خرافة - قول خاطىء
Method	طريقة
Modularity	قابلية التجزئة
Module	وحدة برمجية
Monitoring	المراقبة
Management	الإدارة
Multiprogramming	البرمجة المتعددة

- N -

Need	حاجة
Network	شبكة

- O -

Object	كائن - شيء
Objective	هدف
Organizing	تنظيم
Object Oriented Programming	البرمجة الكائنية (أو الشيئية)
Operation	عملية

- P -

PERT: Project Evaluation & Review Technique	طريقة تقييم ومراجعة المشاريع
Phase	مرحلة

Planning	تخطيط
Polymorphism	تعدد الأشكال
Process	عملية - معالجة
Product	منتوج
Prototype	نموذج تجريبي
Pseudocode	- شبيهة الشفرة
Performance test	اختبار الأداء
Project	مشروع
Project Manager	مدير المشروع
Project Management	إدارة المشاريع
Procedure	إجراء

- Q -

Quality Assurance	تأكيد الجودة
Query	استعلام

- R -

Resources	مستلزمات - موارد
Recovery	تعافي - رجوع للحالة الطبيعية
Requirements	متطلبات
Reliability	الاعتمادية - بدون عطل وخلل
Risk	مخاطرة
Review	مراجعة
Release	إصدار
RAD(Rapid Application Development)	إعداد سريع للتطبيقات

- S -

Stakeholders	ذوي العلاقة بالمنظومة

Scope	حدود
Spiral Model	النموذج اللولبي
Software engineering	هندسة البرمجيات
Software	برمجيات
Source code	الشفرة المصدرية
Stamp Coupling	الاقتران الطابع
Specification	مواصفات
Structured Query Language	لغة الاستعلام البنيوية
Standard	معيار
Structure chart	مخطط هيكلي
Structured programming	برمجة هيكلية
Stress test	اختبار الضغط
Staffing	التوظيف
Stub	وحدة برمجية مؤجلة الإنجاز
Scenario	سيناريو
Securitry	الحماية

- T -

Technology	التقنية
Task	مهمة
Technique	طريقة فنية
Tool	أداة
Testing	اختبار
Team	فريق
Transaction	معاملة

- U -

Unit test	اختبار الوحدة

Usability	سهولة الاستخدام
User interface	واجهة المستخدم
User manual	كتيب المستخدم
Use Case	حالة استخدام

- V -

Validation	المصادقة
Verification	التحقق
Visual Programming	البرمجة المرئية
Visual Basic	بيسك المرئي
Version	نسخة

- W -

Walkthrough	مراجعة
Work Breakdown Structure	مخطط هيكلية تجزئة العمل
Web site	موقع وب
Waterfall Model	النموذج التدفقي- الشلاّل

ملحق (2)

قاموس أوائل الحروف

Glossary of Acronyms

ACM	Association for Computing Maschinary
CASE	Computer-Aided Software Engineering
COCOMO	Construction Cost Model
CPM	Critical Path Method
DBMS	Data Base Management System
DFD	Data Flow Diagram
DSS	Decision Support System
DD	Data Dictionary
ERD	Entity Relationship Diagram
ERP	Enterprise Resource Planning
ES	Expert System
GUI	Graphic User Interface
HW	Hardware
HIPO	Hierarchy Input Process Output
IBM	International Computing Machinary
IS	Information System
IT	Information Technology
JAD	Joint Application Design
JRP	Joint Requirement Planning

QA	Quality Assurance
QC	Quality Control
OO	Object-Oriented
OOA	Object-Oriented Analysis
OOAD	Object-Oriented Analysis and Design
OOD	Object-Oriented Design
OS	Operating System
PERT	Project Evaluation and Review Thecnique
PM	Project Management
RAD	Rapid Application Development
SW	Software
SQA	Software Quality Assurance
SQL	structured query language
SDLC	Systems Development Life Cycle
UML	Unified Modeling Language
VB	Visual Basic
WBS	Work Breakdown Structure

ملحق (3)

تعريفـات

Glossary of Terms

- **Actor:** Anything that interacts with the system.
- **Agile Methodology:** Selection from a variety of suitable tools and techniques to best perform the tasks at hand. It balances between Productivity And quality.
- **Brainstorming** :A technique for generating ideas by encouraging participants to offer many ideas in a short period of time.
- **Controlling** is all managerial activities that are done to ensure that the actual operations go according to plan
- **Class Diagram**: A graphical Representation of a system static object structure، showing classes of the system and their relationships.
- **Class**: A set of objects that share the same behavior and attributes.
- **Constraint**: limitation that may limit a solution or a process.
- **Context DFD**: An overview diagram that is used to show the scope of the system، or show the system with its environment.
- **Data Flow Diagram(Bubble chart)**: An analysis tool used to depict the flow of data through a system and the processing performed by the system . Its Abbreviation is: (DFD).
- **Data**: Raw facts about people، events، things، and places that is important to the organization.
- **Data modeling**: A technique used to model business data requirements and database systems.

- **Data requirement:** A representation of user's data in terms of entities، relationships، attributes.
- **Database:** collection of interrelated files(tables).
- **Decomposition:** The act of breaking a system into components.
- **Directing** is all the managerial activities that are done to lead the subordinate efficiently and effectively in order to obtain the objectives using motivation، communication and leaderships
- **Documentation:** An activity of recording data and specification of a system for current and future uses.
- **Entity:** A class of persons, objects, places, or concepts.
- **Entity Relationship Diagram (ERD):** An analysis tool used to depict data items of entities and relationships.
- **Estimation:** Assessments of the costs, time or effort required for system development.
- **Feasibility:** The measure of how beneficial the development of a system would be to an organization.
- **Functional requirement:** A description of services and activities a system must provide .
- **Gantt chat:** A chart(bar chart) used to depict project tasks against time durations.
- **Information technology:** The combination of hardware, software with telecommunication (networks).
- **Interview:** A face to face fact-finding method of collecting data about the system under development.
- **Joint requirement Planning (JRP):** The use of workshops(sessions) to bring together (stakeholders): customers, users, analysts, designers, and programmers to jointly analyze problems and define requirements..
- **Knowledge:** Data and Information that is further refined based on the facts, judgments, and expertise of the recipient.
- **Milestone:** An event indicating the completion of a major project deliverables.
- **Model:** A representation of a reality, usually in the form of a picture.
- **Organizing** :Activities that result in the design of the structure of tasks and authorities

- **Observation:** A fact-finding method(technique) wherein the systems analysts participates in or watches a person perform activities to know more about the system under study.
- **Opportunity:** A chance to improve the organization even in the absence of an identified problem.
- **Planning** is all the managerial activities done to define the objectives of a project and the ways (project plan) to obtain these objectives.
- **Procedure:** Step-by-step set of instructions and logic for performing a business process.
- **Project initiation:** An activity where by: business scope، goals ,schedule، and budget are defined.
- **Process:** Work performed by a system in response to incoming data flows or conditions.
- **Project:** A sequence of activities that must be completed on time ,budget، and according to specification.
- **Prototype:** A small scale، incomplete، but working sample of desired system.
- **Project** is set of related activities strive to reach a goal
- **Project management** is a system of rules، conventions، practices، methods and knowledge that provide the planning، controlling، directing، organizing and staffing necessary to manage a software project to a successful conclusion
- **Project management:** «the study and use of tools، methods، rules، conventions and practices in planning، controlling، directing، organizing and staffing necessary to manage a software project to a successful conclusion
- **Project manager:** He is the one who is responsible for the success of the project، and has the authority and accountability over the whole project.
- **Review** is a quality assurance mechanism by examining all or part of software under development for the purpose of finding errors
- **Requirement:** A Sw capability needed by the user to solve a problem or Achieve an Objective.
- **Sampling:** The process of collecting a representative sample of documents، forms، and records.
- **Scope:** The boundaries of a project.